Detection of application used on a mobile device based on network traffic

Dmitri Kiritchenko

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.5.2018

Supervisor

Prof. N Asokan

Advisor

D.Sc.(Tech) Mikael Mohtaschemi

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Dmitri Kiritchenko

Title Detection of application used on a mobile device based on network traffic

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Communications Engineering **Code of major** ELEC3029

Supervisor Prof. N Asokan

Advisor D.Sc.(Tech) Mikael Mohtaschemi

Date 31.5.2018 **Number of pages** 70 **Language** English

Abstract

Smartphones have become very popular over the past years, thus being owned by almost every individual, the devices also follow their owners throughout the day thus having access to a lot of information about their users. Additionally various companies provide additional services through applications on mobile devices which makes them highly interested in what people do with their mobile devices, as it allows perfection of these services.

To collect usage data, on top of having user consent, a company must be able to actually see what is happening on the device. But in regards to growing concern about user privacy, operating systems on mobile devices isolate applications limiting their access to only a small part of information of what is happening on the device. Options like running surveys exist, but are highly dependent on honesty of the people and expensive.

To gain the information about running applications network traffic can be utilized as more and more devices are constantly connected to the internet. On the other hand, as well as application isolation, the network traffic is also being more and more protected.

This thesis starts with reviewing previous works to give a picture of what kind of information can be extracted from mobile device and it's network traffic and how it can be used. The main aim of this thesis is to implement a system that detects the used applications and their running times by combining mobile network traffic with application launch times and using machine learning. To assess the detection quality and scalability thoroughly, several tests are performed.

The implemented detection system shows good potential as it achieves near perfect results in optimal conditions, yet to provide these conditions in every case, a lot of work has to be done still.

Keywords Feature extraction, Internet, Market analysis, Random Forest, Smartphones

Preface

I want to hugely thank my instructor Mikael Mohtaschemi for the provided awesome guidance throughout my thesis. Additionally I want to thank all of the people at Verto Analytics for providing me with required infrastructure for realizing the thesis and ideas of what can be done in difficult moments. I would also like to thank Professor N Asokan for his open mindedness, quick reviews and well explained comments and guidance. Also big thanks to my family for supporting me throughout my studies and making it possible for me to be where I am today.

Otaniemi, 31.5.2018

Dmitri Kiritchenko

Contents

Symbols and abbreviations

Symbols

A	Application runs set
C	Classes set
F	Flows and features set
G	Vector of class probabilities
P	Set of all flows with their respective features
P_r	Flow features related to application run r
P_r^*	Filtered flow features related to application run r
α	Accuracy function
β	Badness metric
δ_{xy}	Kroneker delta function of x and y
ϵ	End Time function
η	Skew
γ	Classifier
κ	Kurtosis
$\omega(t, g)$	Occurance count of t truth class index and g predicted class index
ϕ	Precision function
ψ	Mean absolute deviation
ρ	Recall function
σ	Variance
θ	Number of selections to test
υ	Ratio of excluded applications
ξ	Start Time function
a	Number of all applications
b	Predicted application by probability sum
c	Class index
e	Number of excluded applications
f	Flow index
h	Predicted application by thresholded probability sum
o	Predicted application by probability sum
r	Application run index
s	Number of selected applications

Operators

:	Logical if
$[x_1, .. \ x_i]$	Vector or ordered set of i values
$\bar{X}$	Mean of values in X
$\wedge$	Logical and
$\lfloor x \rfloor$	Biggest integer value that's smaller than x
argmax(x)	Index of maximum value of x
ln(x)	Natural logarithm of x
maxcount(x)	Count of maximum value's occurance in x
max(x)	Maximum value of x
$\sum_i^n$	Sum over index i running from 0 to n
$\sqrt{x}$	Square root of x
$\{x_1, .. \ x_i\}$	Set of i values
$x \in X$	Value x belonging to set or vector X
$\mid$	Logical for
$\lvert X \rvert$	Count of values in vector or set X
$\lvert x \rvert$	Absolute value of x

Abbreviations

ACK	Acknowledgement Number
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CPU	Central Processing Unit
CSV	Comma-separated values
DDCasino	DoubleDown Casino
DES	Data Encryption Standard
3DES	Triple DES
DPI	Deep Packet Inspection
HMAC	Keyed-Hash Message Authentication Code
IKEv2	Internet Key Exchange version 2
IP	Internet Protocol
IPSec	Internet Protocol Security
MAC address	Media Access Control Address
NAT	Network Address Translation
OS	Operating System
P2P	Peer-to-peer
PC	Personal Computer
RAM	Random Access Memory
RAN	Radio Access Network
SDK	Software Develompent Kit
SHA1	Secure Hash Algorithm 1
SSH	Secure Shell
SVC	Support Vector Classifier
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VPN	Virtual Private Network
WZ	Wizard of Oz Slots Free Casino
Wi-Fi	Wireless Fidelity (Wireless local area network)
XOR	Exclusive or operator

1 Introduction

Mobile phones are nowadays following their owner's every step and store massive amounts of personal information. A lot of effort is put simultaneously into protection and exploitation of owner's privacy, which can be seen in many examples given by Tourani et al. in [1]. The information from mobile devices is very valuable, as it is very personal to the device owner and is usually not faked. For example big social events or spread of sicknesses can be predicted by looking at the movements of people as shown in [2–6]. Trends and preferences of people can help market prediction and thus provide better ways for growth for companies. On the downside personal information can also be used for less ethical actions, like hackers gathering information about people for blackmailing, phishing or targeted hacking purposes. Additionally countries can spy on their own people to for example detect anti-government individuals and monitor their activity more closely or change the content people see while browsing the internet as shown in [7, 8].

Because of the vast data availability, companies and individuals try to protect mobile devices from misuse of available information, as users do not always get notified properly about gathering of this information. This protection is mostly performed by isolating applications inside the mobile device and setting very strict policies on what can be accessed by each of the installed applications. With isolation strength growing day by day, the information gathering from within the mobile device becomes more difficult. Although there are still many things that can be said about mobile device users by looking at vast sensor data provided by the device, there is information that cannot be gathered that way.

Aside from collecting information from inside the mobile devices, there are options to collect it from mobile operator's infrastructure or the internet. This way no special permissions from the device itself are needed, but instead there has to be access to the path over which the devices communicate with the world. For example it is not hard for operators to gather information about which mobile network radio base stations a specific device connected to over time. Already this knowledge can give a lot of information about a specific person's life and habits as shown in [9, 10]. The base station logs, however, are usually only accessible by mobile connection operators who might not be eager to share this kind of data easily.

To get more fine grained information about users without having access to their mobile devices, one can examine the network traffic associated with the devices. Because devices are almost constantly connected to the internet, one could potentially tell what user is doing with his/her mobile phone throughout the day. For example one could look at the amount of data transferred during different times by the device and try to infer what kind of things the person likes to do by defining activities, like watching videos, listening to music or reading news. The level of detail of this data can be made even higher by having a look at the endpoints of the connections related to the device, which could potentially give exact service names the user prefers. Information gathered this way allows targeted advertising or enables companies to make business decisions based on current trends as discussed by Hautala in [11].

To limit the availability of this fine grained data about users, there are options like

Virtual Private Network (VPN) for keeping the connection between the mobile device and a VPN Server encrypted and secured. Additionally many applications encrypt their traffic to protect user's privacy. This disallows peeking into the transferred data itself, but it is still impossible to avoid third parties to see the data transfer itself.

This thesis aims at extraction of information about mobile device to identify user habits. Extraction is performed by looking at encrypted data transfers related to a mobile device. As the encrypted data does not provide much to look at, the amount of usable information is relatively small. To get the most out of it, a known machine learning algorithm is applied to analyze the available data and make predictions of actual events.

1.1 Motivation

Market analysis has played an important role in perfecting and optimizing products and marketing. For example companies can check how changes in applications reflect on the market in cases of their own changes as well as their competitor's changes. Instead of spending people's time on surveys, which also give a lot of room for cheating, a good approach for gathering this data is following the mobile phone usage. Especially as more and more people are moving from owning personal computers to having phones and tablets which follow them throughout the day.

However ever more often using of strong encryption for data transferring and having a secure OS design is emphasized. As a result current mobile operating systems are constantly strengthening application isolation, thus closing off possibilities for information gathering even in cases where users consent to the gathering themselves. A part of information being hidden is application usage on a mobile device with exact knowledge of which application has been used and how much. Therefore an approach able to get this information with system's restrictions in place is needed.

1.2 Author's role

For the work presented in this thesis the initial state was a working VPN setup together with test devices with installed application for gathering foreground information. Additionally several ideas have been proposed by authors coworkers. The author carried out the development and evaluation of the automated application detection system in following aspects:

- Configuration of traffic capture on VPN server
- Automation of application launching on test devices
- Feature extraction and alignment for ground truth
- Utilization of machine learning algorithms
- Configuration for bad connection tests
- Manual usage tests

1.3 Structure of the thesis

The first part of the thesis will consist of descriptions of relevant techniques in Sec. 2 and a literature review in Sec. 3 of works related to retrieving and utilizing information gathered from mobile devices. Starting off by looking at call and usage information, which has been available for longer time and then continuing into rapidly growing network usage's information gathering.

The second part of the thesis focuses on solving the problem of application recognition. In Sec. 5 and Sec. 6 the solution to the problem is reasoned and described thoroughly. Sec. 6 additionally describes the metrics used to understand how well the approach works and highlight what are the actual problems.

Finally Sec. 7 goes through various detection quality analyses made with the implemented system and finds explanations to presented results. These explanations show how the system might be improved in terms of detection performance. Various improvement ideas and their implications are discussed in Sec. 8 together with understanding of limiting factors for applying this method in production environment.

2 Background

As this thesis is highly involved with network traffic interception, it is good to understand what stages does the mobile device's connection to the internet actually involve. This section provides an overview of the connection to the destination servers showing the possible data capture points en route. The network structure information is followed by descriptions of various algorithms used in different parts of this thesis.

2.1 Network structure

This section shows and describes various points involved with network traffic transport together with their potential access to the relevant network traffic information.

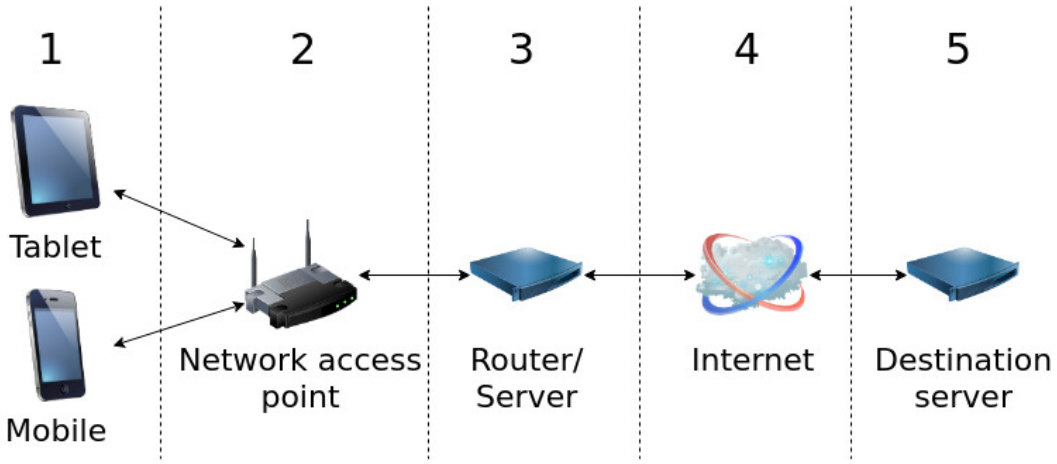


Figure 1: Network stages for data capture

There are five distinguishable points at which the network information could be gathered as shown in Fig. 1:

1. the mobile device itself. Here the data could be captured by an application. As usually it is impossible to see all network traffic from within an application, an alternative to it, is having a virtual network interface within the mobile device and for example use VPN to force all of the available traffic through this virtual interface.

2. the first network node that the device connects to, usually a local Wi-Fi or a radio access network (RAN) access point of an mobile operator. It sees all of the network packets going in and out of the phone and additionally can utilize radio signal information.

3. the server through which the connection of a mobile phone is forced. The enforcement can be done by configuring operator's routing systems or by a VPN connection.

4. all of the internet. Usually it is almost impossible to securely get all the packets from a specific phone at this stage. Instead of looking at a specific device it is possible to do a general study of the traffic going through a link you have access to.

5. the server that a specific connection from the phone points to. The network statistics at this point can be utilized by an application developer for example to make better services. But mostly at this point one can only see the traffic from own applications.

Most of the studies are executed on points 1-3. Examples of these studies shall be presented with their results and and use cases in Sec. 3.

2.2 VPN and encryption

Over the topology shown in Fig. 1 a VPN can be used to connect stages 1 and 3 together. In scope of this thesis VPN host communicates over an IPSec protocol described by Kaufman et al. in [12] and uses IKEv2 key exchgange described by Kent and Seo in [13] during connection initiation.

One effect of this is that from connection's perspective everything between these two stages is considered as a single link and there is a sub network with separate addressing, which makes address tracking much easier. Additionally everything is forced through this link, making sure that all of the needed data will flow through selected server for stage 3. On top of that this virtual link encrypts data it transfers, which disallows eavesdropping over the connection between these two points.

The encryption mechanisms considered in this thesis are Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES), because these happened to be the default selections of the test devices. Technical description of AES can be found in Daemen's and Vincent's technical documentation [14]. And for 3DES the description can be found in documentation by Barker et al. [15]. In case of AES the block size is 128 bytes and in 3DES it is 64 bytes. Because the algorithms are designed to only parse full blocks and the length of the data to be transferred is not always dividable by the block size, padding must be applied to the data.

Padding usually adds some values with predefined rules to the end of the data being sent, which naturally increases the size of the sent data. E.g. in 3DES specification padding consists of values starting from 0 and increasing until the block is full (00, 01, 02...) and to ensure right parsing of the data in cases the data length is divisible by the block size, another block of pure padding is added to the end. [16]

As the encryption algorithms considered in this thesis parse data in small blocks, the same key could be applied many times. This degrades security because for example if same message would be sent several times someone could try to predict which message it is by judging from the context and be able to infer the key from this message. An obvious solution of changing the encryption key very often is not practical. To fix this issue various encryption modes exist, the one considered in this thesis is called Cipher Block Chaining (CBC). CBC goes through data block by block and every time before encryption it XORs previous ciphered block with

the current plaintext block. This way even if the same block is encrypted twice the result will depend on the previous block and probably be different. [17]

Additionally Keyed Hash Message Authentication Code (HMAC) can be used, which in conjunction with encryption provides authenticity, integrity and secrecy of sent messages. More information on HMAC, can be found in Turner's documentation [18]. A lot of functions exist for calculating the hashes for HMAC, for example Secure Hash Algorithm 1 (SHA1) is used in this thesis. The output of the hashing function can also also truncated for example to 96 bytes for bandwidth savings. More specifics on full encapsulation procedures can be found in Dorsway's and Madson's documentation [19].

2.3 Machine Learning

One option for utilizing of parsed data from VPN is to feed it into machine learning algorithms. Two algorithms are considered in this thesis, namely Support Vector Classifier (SVC) and Random Forest.

SVC is considered to be a good machine learning method of data classification, and is based on setting data to hyper planes, defining a kernel for splitting of the data and applying the split on data points being classified. A description of the method with examples and implementations is given in a guide by Hsu et al. [20].

Random Forest on the other hand, is originally described in Breiman's article [21] and is an extension of Bagging method from his earlier article [22]. It is a decision tree algorithm, meaning that during the training phase, the algorithm comes up with a set of consecutive if clauses which can be visualized like a tree. Starting from the root of the tree, the process systematically descends by selecting the matching answer at each node. The process ends in a leaf of the tree which states the predicted class for given data. Having only one decision tree is usually not enough, because the data is not always very consistent and extracted features may vary between different data points of one class. Having multiple different trees allows looking at the data from many perspectives and thus making better decisions.

One of the biggest problems in generating multiple trees is that while training them on a same set of data the resulting trees might be very similar or even duplicates of each other. Thus different methods are applied to ensure that every new tree brings some value to the classifier. A method considered in this thesis includes two levels of randomness. Firstly each new tree is trained on a random subset of ground truth data, this is called Bagging. In addition to that Random Forest uses only a randomly selected subset of features on each node. This adds more variation into the trees as each node does not see the whole picture and makes the classifier more robust. All in all Random Forests are reasoned by the law of large numbers, which in concatenation with randomness in correct places yields very good results.

3 Literature Review

This section informs about different use cases and possibilities driving the studies related to mobile devices. First looking into general mobile usage statistics that could be gathered for example by mobile network operators, it can be seen that surprisingly many things are deducible from available data.

3.1 Phone event analysis

This section shall handle the information about mobile phone usage, which can be gathered by the mobile network operators. Operators generally keep this information to themselves, as it is quite personal, but can share it in some cases with other parties.

To estimate the wealth of population in different areas Blumenstock et al. in [9] utilize various statistics extracted from mobile phone usage in Rwanda. This approach allowed generating very detailed wealth maps as well as prediction of single individual's status. Wang et al. [10] manage to approximate age, income and even the residence region based only on a graph of interactions between pairs of users.

Another example of using user interactions is predicting the spread of epidemics. [2–4] have studied the spread of infections like malaria and HIV in Africa and noticed features in mobile communications which correlate strongly with infection spread. In [5] Frias-Martinez et al. give an example of utilization of this information in case of H1N1 spread in Mexico and claim that the impact of the virus was reduced with help of their study by 6-10% compared to simulation without intervention.

When limiting oneself only to the locations people have visited, a lot can be told as well. Using mobile devices is more or less a unique way to gather location tracking information, because it is easy to organize, cheap and quite precise and additionally there are no other feasible ways to gather it. Of course the precise location of devices is not always available, but even knowing the operator's transmitter to which the phone is connected at an instance of time gives vast possibilities.

One way of using location information is for example optimization of roads for cars or public transport routes. A good optimization example is presented in [23], where Cici et al. show that sharing cars could drop number of cars in Madrid by 67% if people would not mind having a little longer work trips. Giradin et al. give an example of how to optimize tourist business in [24], by knowing what places they like to visit and when. In [6] Linardi et al. even suggest, that it is possible to predict some violence outbreaks, by looking at societal tension which is reflected by the mobility of people.

3.2 Network usage analysis

As popularity of having constant internet access on mobile devices is growing, it is possible to dig deeper into phone usage to extract more valuable information. For example by looking at the amounts of generated traffic it is possible to optimize the network. In [25] Kumar et al. show that only 40% of base stations are needed most

of the time, which would allow rest of the stations to enter sleep mode and save quite a lot of energy. A study reported by Yu et al. [26] shows that by predicting future networking events it is possible to reduce the energy consumption during inactivity time by 56%. Espada et al. [27] and Shafiq et al. [28] show that different base stations get different kinds of traffic and on top of that the stations with similar kind of traffic are often close to each other, thus allowing to assign applications to places where they are popular. [29] Also it is for example possible to prove, that mobile traffic appears more often from well charged phones and from non-home locations as shown by Soikkeli and Riikonen in [30].

3.3 Detecting fraud by analyzing network traffic

Next when concentrating only on the traffic related to one mobile device it is for example possible to detect fraud in application's advertisement, as some developers prefer to make money on placing advertisements into their apps instead of requiring users to pay. There is a possibility to cheat the advertisement counter to think that the ad has been displayed or even clicked. In [31] Crussell et al. propose a way to detect these frauds by running the app in a simulator and looking at generated connections. [32]

Nowadays malware is also an emerging problem on mobile phones, as the mobile phone user base and the amount of personal data stored on these phones are both large. On top of that phones are almost constantly connected to the network allowing the developers of malware to communicate with their creation and utilize it. Therefore a good approach for detection of malicious applications is by monitoring network traffic and many times information about things like memory and battery consumption are also used. [32]

It must be noted that machine learning is very popular in this scene. Shabtai et al. [33] propose to build predictive models, which after training are able to detect anomalies and tell if the phone has malware installed. The algorithm utilizes a lot of different metrics gathered from the phone, including features extracted from network communication. Su et al. [34] on the other hand, try to prevent malicious applications from even getting onto phones by running the applications in virtual environments and test phones beforehand and feeding their network traffic into machine learning algorithms. Another approach is looking at the connection destinations initiated by the phone, Wei et al. [35] and Zaman et al. [36] both demonstrate that this approach works well as long as mobile telephone's DNS traffic can be monitored.

There are also applications which are not necessarily considered malicious, but send sensitive information about the users into internet. In the worst case, this information is not even encrypted, thus allowing anybody with ability to monitor the network to see this data and even identify to whom it belongs. Collecting this data is not always bad, one use case could be targeted advertisement which would lessen the annoyance for the users and provide better marketing results. Therefore some advertisement libraries collect this information and for example in [37] a study is performed by Kuzuno and Tonami to detect the cases where this information is not transferred securely. [32]

3.4 Device usage analysis

Continuing deeper into network traffic analysis it is also interesting to study how exactly users utilize the functionality of their devices and installed applications. Multiple facts can be deduced solely based on network traffic captured from the phone's connection to internet. For example Ruffing et al. could identify device's operating system in their publication [38], which relies on frequency of transmitted packets. This kind of approach allows 70% accuracy already during first 30 seconds of packet interception from most popular mobile operating systems. When the used application is known and traffic capture is longer the approach reaches near perfect results.

Instead of detecting operating systems, one might want to know which applications are in use currently. Sometimes this includes applications done by other developers so that there is no access to their inner statistics. One of the use cases could be quality of service prioritization based on an application detected from the network traffic itself. For example Lu et al. build a prioritization system in [39], which uses Deep Packet Inspection (DPI) to limit unwanted Peer-to-peer (P2P) traffic in mobile networks so that other users will have enough bandwidth.

In [40] Finamore et al. deduce, that Youtube (an on demand video service) is used very similarly on mobile and non-mobile devices with exception that mobile users finish their videos less often. Furthermore users can be fingerprinted by their habits of phone usage gotten from for example network traffic logs. This enables tracking and detecting individuals to some extent even in cases where user switches the device. In [7] Verde et al. present a machine learning system, which can detect specific user's traffic even when the user is behind a NAT, thus sharing an IP address with multiple other users.

In [8] the encrypted connections of the mobile phones are meddled with in such a way that Rao et al. can see the plain traffic and know exact statistics about it. As the mobile device tries to start an encrypted connection with the destination server, the author's server gets in the middle and acts as the destination server for the mobile device and vise versa. This method is called SSL bumping, it peeks into the actual traffic of the device, thus fully invading privacy. Rao et al. do not present any analysis over time, thus if something changes, the whole system might need to be retrained. Authors also mention that some applications detect the SSL bumping and do not accept the connection, in the future it is possible that more applications will start doing that, thus blocking this approach. Additionally authors show how applications send personal statistics in an unprotected way over the network and analyze how governments and internet service providers can manipulate what users see. A similar result is shown in a study [41], where Alan and Kaur achieve an application detection accuracy of 88% in best case, while having almost 10 times more applications as Rao et al. in the testing dataset. However the paper lacks proper manual usage tests which could show that the method does not actually work well.

The two studies with best results in application identification are mentioned by Conti et al. in [32]. In both studies research groups reach 96% identification

accuracy at best. The difference between these two studies is that in the study by Mongkolluksamee et al. [42] the traffic is captured on the phone itself, while Taylor et al. [43] are tapping on the connection on the network. In both cases Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) headers are required so these methods will not perform well when for example an IPSec is being used by the phone or by an application.

Lastly it is even possible to detect the actions users are performing inside the applications by only looking at network traffic and sometimes even some specifics of these actions. These studies mostly target apps intended for communication and social interactions like instant messengers and social networks respectively. Apps in both of these categories often contain very sensitive information about the users. Works by Coull and Dyer [44] and Park and Kim [45] concentrate on communication category and gain near 100% accuracy on detecting actions with Apple's iMessage on iOS and Korean KakaoTalk on Android respectively. On top of that Coull and Dyer in [44] can even tell which language is being used for communication with over 80% accuracy. Conti et al. show very good results in [46] for such social applications as Facebook and Twitter as well as productivity applications, such as Dropbox, Evernote and Gmail. Overall it is quite scary how many things can be deduced by monitoring solely the network traffic emerging from a mobile device.

3.5 Ground truth acquisition methods

Ground truth generation is the first logical step to approach identification of applications from network traffic. It is important to have a good definition of what can be considered reliable information and what kind of properties and limitations the information could have. On top of that one has to be able to gather this information in a feasible manner, this is especially important when machine learning algorithms are used, as the required amount of data for teaching these algorithms reliably is usually very large.

In [47] Korczyński and Duda propose an application detection based on hostname matching of the IPs the application tries to connect to. Using this matching flows are mapped to specific applications thus teaching the classifiers based on that. Authors mention that sometimes applications connect to IPs, which do not have a public hostname and thus these flows will not be utilized for detection. Another flaw of this approach is that for example one developer's different applications might have connections to the same endpoints, thus making it impossible to distinguish between these apps.

Similarly to Korczyński and Duda [47] in [41] Alan and Kaur collect traffic from a node on the network. But instead of looking at whole traffic dumps, the interest is shifted to the traffic being transmitted immediately after an application is opened. To gather this dataset they use two tablets and two phones and automate application launches on these devices. The applications are selected from 1595 most popular applications in application store and the data gathering is ran for about a month generating about 86 thousand recorded application launches. Launching is executed over the Android Debug Bridge which allows device control over USB cable and

traffic is captured with a Linux program: tcpdump.

Taylor et al. [48] describe in detail their generation of a dataset containing user actions. They utilize a tool called monkeyrunner to automate user actions within the studied applications and record the traffic on a network node saving the data into CSV format. Each data point corresponds to a received packet and contains information about the reception time, involved IP addresses and ports, packet's size, protocol and in case of TCP, the protocol's flags. Authors also propose a split of traffic into bursts which are defined as time windows of set length and multiple intercepted packets. These bursts are the ones being assigned to applications and thus allow detection of app in near real time. Authors also mention the possibility of simulating the devices for generating data thus enabling scalability of the method.

Alternatively in [49] Wang et al. place their collection point into an encrypted Wi-Fi environment. In their approach, the captured traffic is split by MAC addresses to identify the devices. The results show that for example it is possible to make a sophisticated prediction of the device's operating system. On the other hand, the study is done in information security context and the authors were not interested in inferring a large variety of applications. They composed a training set of a very few apps and ran and annotated by hand to get these results.

Instead of automating data generation, Le et al. [50] propose a crowd sourcing approach in which an application generating ground truth data is given for users to install. On top of gathering ground truth, the application provides services like detection of privacy leaks and freedom of selection of which applications can be monitored thus giving users some gain from installing the application. The author's application modifies the outgoing packets inside the mobile device by adding the application that generated this traffic into the packets. The traffic is then routed trough the author's VPN server, which extracts the required features from the transmission and forwards it to the internet. Additionally authors also think about the nature of different network protocols and to have the test setup as close to reality as possible they use UDP in their VPN tunnel, because it resembles IP's behavior. A similar approach is introduced by Iwai and Nakao in [51], where instead of having an application, modified smartphones are handed to people. The modifications also add the running application into the traffic flows and the data is again intercepted on a VPN server.

3.6 Detection techniques

With available ground truth a suitable algorithm has to be selected, to utilize the data and perform the classification.

3.6.1 Markov chain based approaches

Korczyński and Duda propose in their work [47] a Markov chain based approach, which has discrete time states and transitions between these states are defined randomly with certain probabilities. Transmission control features are extracted from packets headed away from the user, these features are enumerated and considered

as states mentioned above. It is also possible that multiple control features are embedded in a single packet, thus creating additional states with combination of these features. During traffic analysis these states are found and the probabilities of moving from one state to the next are calculated by the relative number of occurrences of this transition. This also requires two special states that are not extracted from packets: an entrance state and an exit state. These two states behave much like the others, except for one cannot re-enter entrance state or return from exit state.

By having a model like this one can create a metric for each captured set of packets, which shows how often each state is visited during the connection. The Markov chain model also allows calculating the probabilities of each state thus enabling easy predictions of which application is actually the one generating this traffic. [47]

Shen et al. [52] extend the Markov chain method by introducing second-order Markov chains, which calculates the distribution for the next state based not only on the current state but also on the previous one. Because for each transition a knowledge of two previous states is required, the entrance state for the chain embeds probabilities for the next two states. This way when we take the next step after entering, we know the required states for future calculations.

3.6.2 Machine learning based approaches

Instead of creating highly customized models it is possible to use general approaches for classification. For example in [50] Le et al. used SVC and got results ranging in accuracy from 73% to 96%. These are good values but overall SVC tends to be much more computationally expensive than Random Forests discussed next.

A good example of the difference between SVC and Random Forest is given in [48], where Taylor et al. notice that Random Forests are almost twice as fast and require half of the disk space compared to the SVC, while giving better classification results. Therefore the Taylor et al. used this algorithm in their following work [43] and achieved good results. Conti et al. used this approach in [46] as well. Additionally Mongkolluksamee et al. [42] mention that Random Forests includes information about feature importances by default, which are very useful for analysis.

4 Problem statement

It is very important for many companies to know how people use applications and mobile devices in general to gain better knowledge of their target audience. With the strengthened mobile application isolation allowing to see only the app start and stop events, new ways must be found to get this information. Additionally applications are encrypting their newtork traffic more often, thus there is no use of peeking into the transmitted packet contents.

However it is still possible to find out application start and stop timestamps and the device's network traffic which can be captured over a VPN server. Thus the goal is to implement a classifier suite which would detect the application currently opened on a mobile device accurately. As there are already works with machine learning based approaches for detection of applications based on network traffic, an extension on such approach should be presented with utilization of application start and stop timestamps and provide better results.

To create and evaluate the classifier, the following goals must be met. There is a need for a proper automated environment which would generate enough ground truth data for testing purposes. The environment should be able to generate trustworthy data with minimal human intervention to show that the approach can be scaled up and automated properly.

Additionally the scalability requires evaluation of storage and processing requirements to see if the system could handle many devices simultaneously. A cross-device tests also need to be performed to see if there is a need to train on every available device separately. The ability to filter the non-trained applications should also be assessed, as it is difficult to have a classifier trained for all of the applications on the market continuously.

As scalability issues are handled the application detection quality needs to be assessed in several ways, to not only know how many percent of application launches are detected right, but to also see how various applications are cross-detected with each other. The cross-detection studies should allow seeing if there are logical explanations to the problems between particular applications which could be addressed in the future.

It is also important to evaluate the system in non-optimal situations to assess how it would work in reality. Naturally manual device usage should be tested as it is the actual usage for the product created in this thesis. Also very bad network conditions should be tested as the detection is aimed at mobile devices, which tend to move a lot and naturally can have problems with network connection.

Overall the result of these goals should provide a good insight into feasibility of using this approach in a real production environment.

5 Design

Before beginning the work, potential ways for solving the problem have to be selected. Considering the ideas presented in Sec. 3.5, the following decisions were made. In scope of this thesis the crowd sourcing approaches are infeasible for the difficulty of finding enough people to generate enough data in a relatively short time span. Additionally there are no good methods for being sure that these people will not cheat or make mistakes during data generation, if they have to do something manually.

In context of this thesis the hostname matching approach is considered unreliable and cannot be used as ground truth, as the number and variety in studied applications is higher than in Korczyńskis and Dudas work [47]. Therefore an approach similar to Alans and Kaurs study [41] is selected in this thesis. As the approach studies only the application's start time traffic, it is simple to implement and provides very high confidence in detection.

As for algorithm selection, following options were considered. Because encrypted network traffic is used, it is impossible to extract anything sensible from the captured packet payloads. Therefore for example DPI approaches cannot be utilized, as these rely on the ability of reading the payloads. Thus with in this case only the packet headers can be used.

The Markov chain based solutions presented in Sec. 3.6.1 are very elegant, but as this thesis works on encrypted channels the connections look much more monotonic from the outside of the channel. Although it could be possible to deduce communication's progression based on the limited information given by packet headers, it would probably not work very well, as the seen flags are not necessarily reflecting the characteristics of the encrypted data. Additionally the detection rates given by the works that utilized Markov chain approach were not as good as the works with machine learning approaches.

The Sec. 3.6.2 shows, that Random Forest is noticeably more popular and provides better performance compared to SVC. Other algorithms of higher complexity shall not be used for detection purposes in this thesis as for example Neural Network based algorithms could potentially perform better, but require a lot more learning data than the feasible amounts in scope of this work. Additionally as the ground truth data might contain some impurities, it is important that selected algorithm can handle that to some extent. Thus Random Forest is the algorithm of choice in this thesis, as it provides resilience to small impurities in the data, is easy to configure and naturally gives information about detection quality.

6 Implementation

This section handles the configurations of the devices involved in this thesis and the algorithms used for application detection as well as calculation of metrics. Two devices have been used in this thesis and are referenced as a phone device and a tablet device, additionally manual tests in Sec. 7 include a second tablet device with newer OS. The second tablet is identical to the first one in scope of configuration and systems, and is featured in only one test, where the actual differences will be described. While the implementation is tested on these particular devices, the proposed approach is generic as it can be used with almost any device connected to the internet.

First the devices and their configurations will be explained, which is followed by the definitions of terminology and data gathering and extraction used in this thesis. Then the actual data generation and parsing process is explained, followed by the evaluation metrics definitions.

6.1 VPN and device setup

The traffic capturing in this thesis is done on a remote server through which all mobile device's traffic has been routed with VPN which is configured according to the Sec. 2.2 section on a strongSwan [53] based server. strongSwan manages routing tables through iptables [54].

To focus only on the interesting packets, the capture points are set within iptables with nflog, which allows specific definition of logging point during routing process. The actual capture points are in the mangle table at INPUT, OUTPUT and FORWARD points. At these points VPN packet addresses have already been changed to VPN's subnet range 10.0.0.0/17, thus tracking IPs of devices is much easier. There are also following filters applied to discard useless data:

- For INPUT source is from VPN's subnet
- For OUTPUT destination is to VPN's subnet
- For FORWARD source or destination is from VPN's subnet

The tunneling algorithms of the test devices as shown by the strongSwan's status command are AES_CBC_128/HMAC_SHA1_96 for the phone device and 3DES_CBC/HMAC_SHA1_96 for the tablet device. In this case HMAC configuration is the same on both devices, so it should not affect detection performance. On the other hand, the encryption algorithms differ and especially the difference in block sizes as explained in Sec. 2.2 should greatly affect the application detection performance, as the detection relies on packet sizes which in place are depending on padding lengths.

With VPN all of the mobile device's connections are forced through the VPN server The server also can thus save the captured packets using tcpdump into .pcap files. These files are named with the capture start timestamp for ease of use.

Additionally mobile devices have a program installed, that collects data about application launches including application names and exact launch and close timestamps. The program is intended to run constantly in the background, in addition to generation of launch logs, it also tries to keep VPN connection running. This data is very important for automation of application running and mapping it to the network captures.

6.2 Flow definition

After the network traffic is captured it is parsed into networking flows from now on referred to just as flows, which are defined as a group of packets related to one transmission instance between two endpoints on the network. Applications create flows when they want to send or retrieve data from the network. Splitting into flows aids traffic classification, as flows related to specific application launch are expected to start and end within the application's running time. Therefore if there are flows that started before the application opened or after it closed, the probability of this flow belonging to some other application is high. The separation of flows is done mostly via a quadruple consisting of network addresses and ports of both endpoints. Flows can contain longer exchanges for example in cases where data access requires authentication or the objects transmitted depend on previous ones. In some cases previous flows can also be continued so that no new handshakes are needed, this however is not considered in scope of this thesis.

The focus is only on TCP and UDP protocols as these are most frequently applied for application data transmission. UDP, being a stateless protocol, does not reveal information about connection state without looking further into packet contents. Therefore UDP flows are mostly determined by the address-port quadruple, but on top of that there is a timeout of one minute used in this thesis. If there are no UDP packets with given quadruple for the timeout duration, the flow is considered closed and in case new packets with same address-port combination emerge, these are considered to be a part of next flow.

Similarly to UDP, TCP flows also have a timeout, but in this case the value is set to five seconds, as TCP is a stateful protocol. The statefulness means that the flow constantly has a state such as established, listening or closing. During waiting TCP engine usually sends keep-alive messages, which are naturally seen in the traffic. On top of that TCP keeps track of transmitted and received packets, thus one can tell if the data actually reached the other end. This gives a good way of following the flows from initialization until close.

The actual flow extraction is done by Bro Network Security Monitor [55] as it allows custom scripting and can go through captured files very quickly. On top of that flow extraction tries to avoid looking at flows with irregularities to keep the data clean for machine learning. The irregular flows are detected by Wireshark's terminal version tshark [56], which provides an easy to use TCP protocol dissector. The filtering removes flows which contain one of the following:

- TCP retransmission, which happens in case some of the packets did not reach the receiver and are requested to be resent.

- TCP duplicate ACK, which happens in case when the sender did not receive an acknowledgment of reception and to speed things up sends the same data again.
- TCP reset flag, which indicates that something in the connection did not add up and the whole connection should be dropped.

6.3 Feature extraction

To utilize machine learning one must define and extract numeric features from the flows described in Sec. 6.2. The encrypted data in the packets seen on the wire does not give much information for extraction. Additionally time related features like packet inter-arrival times cannot be used, as varying network conditions would mess up the features easily. Thus this work focuses only on packet sizes, direction and order of arrival.

The separation into flows is useful in this situation, as just looking at all of the packets related to one device will result in a lot of noise. For example applications might share some libraries which generate similar traffic and separation into flows will allow locating these common patterns, thus giving them lower priority during the detection phase.

Feature extraction generates in total 70 features per flow, yet some of the features might be left blank in case it is not possible to calculate these values. As we are getting both TCP and UDP flows, this is one of the features used and it is named 'proto'. Next, the packets inside the flow are grouped in three ways, depending on their direction: packets that originate from device, packets that are sent to the device and all packets together. These are reflected as prefixes 'out_', 'in_' and 'total_' respectively. This grouping can help distinguish cases where for example mobile devices request going to the server is always the same, but the server's answers may differ.

From each of these groups packet count is extracted as a feature and named as count. On top of that following features are extracted from packet lengths in captured order: average value, skewness, variance, standard deviation, mean absolute deviation, kurtosis, 11 quantiles ranging from 0 to 1 and autocorrelation. Their nicknames are 'len_avg', 'len_skew', 'len_var', 'len_std', 'len_mad', 'len_kurtosis', 'len_quantile_i', 'autocorr_i' respectively where i is the number of quantile or the correlation shift index.

Following equations show how feature values are calculated. X_i is the i-th measurement, $\bar{X}$ is the average measurement value and $n = |X|$ is the measurement count. The skew in this thesis is calculated with

$$\eta(X) = \frac{\sqrt{n(n-1)}}{n-2} \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^3}{(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2)^{3/2}} \tag{1}$$

For kurtosis, the equation

$$\kappa(X) = \frac{n^2 - 1}{(n-2)(n-3)} \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^4}{(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2)^2} - 3\frac{(n-1)^2}{(n-2)(n-3)} \tag{2}$$

is used. And the standard deviation is calculated by

$$\sigma(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1} \tag{3}$$

in this thesis. Standard deviation is a square root of variance. Both are calculated with delta degrees of freedom set to one. Mean absolute deviation is calculated with

$$\psi(X) = \frac{\sum_{i=1}^{n}|X_i - \bar{X}|}{n} \tag{4}$$

Quantiles include the deciles, min and max of given set of packet lengths. Deciles are selected position-wise from a sorted set of given values. In case the position falls between data points, the value is taken from the linear interpolation between the two data points.

The aforementioned features are basic statistical features, that represent the data in a well defined vector, making it suitable for machine learning. However these features do not capture information about the order of arrival, which is a good way of separating conversations. For capturing information about actual order of the arriving packets autocorrelation is used. The implementation uses convolution theorem, which states that multiplication in frequency domain equals to convolution in time domain as shown in Andrews' and Phillips' book [57]. First the deviation of each of the packet sizes is taken by substracting their mean from them, resulting vector is then Fourier transformed. The absolute values of the transformed vector are squared as autocorrelation is a convolution of the vector to itself. The resulting vector is then inverse Fourier transformed and normalized by the sum of squares of deviation values. Lastly a vector of the size of the original sample vector is taken from the Fourier inversed data starting from the left side. As there are four autocorrelation features for each vector, they are selected as evenly spaced samples between the start and the middle of the autocorrelation vector, because autocorrelation is symmetrical. In case the number of values is not enough to fill the four samples, the samples are filled with zeroes. Naturally when the number of samples is not enough to do autocorrelation at all, the feature values are set to zero.

In TCP flows it is also possible to predict the packets, that only acknowledge reception without carrying any data, by defining their length to be 52 bytes. This creates a feature of ratio of acknowledgment packets in comparison to total packet count, which could potentially show how single sided a TCP connection is and is noted as 'just_ack_ratio'.

6.4 Ground truth

As the infrastructure is defined, it is possible to define the actual data gathering processes. Solutions and decisions for ground truth generation were selected because they were the easiest and the most straightforward at the time of implementation.

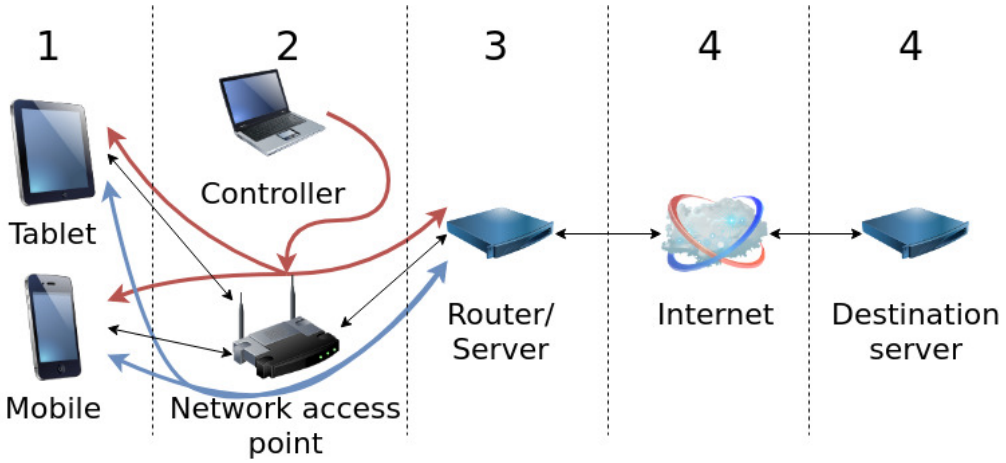


Figure 2: Location of controlling PC in the whole network structure. Red lines denote the paths used by controller for coordination by sending launch and capture commands. Blue lines denote the path of encrypted VPN traffic which gets captured for analysis.

The data generation process is coordinated by a Linux PC which sends commands through Secure Shell (SSH) to the mobile devices and the server over the paths denoted in the Fig. 2 with red lines. Channels of communication between the coordinator and the devices are separated from the blue colored capturing path of the ground truth packets, thus having no effect on the data.

The sequence of the commands is following, first stop all running applications on the device and the network capture on the VPN server and wait for 5 seconds. Next start new network capture and wait additional 5 seconds to be sure it has really started. Then a randomly chosen application is opened on the mobile device and after 20 seconds, the application is killed. The process continues from the start of this paragraph until stopped. On process stop the running application is killed and the network capture is stopped.

After the data generation process has been running, the ground truth data can be collected from the mobile devices and the VPN server as mentioned in Sec. 6.1. From the VPN's pcap files the flows are extracted according to the process described in Sec. 6.3. The real application launch data is also fed into the alignment engine. For each application start timestamp given by mobile device, this engine finds the pcap file with the highest timestamp which is still smaller than the opening timestamp. In that file, each flow's start and stop timestamps are checked against application open and close timestamps and if the flow is fully within the application running time it is considered a ground truth datapoint. These datapoints are then saved with their respective application for machine learning.

6.5 Machine learning

As enough of the ground truth data has been gathered, it is possible to utilize it for training and testing the application detection algorithms. Scikit-learn [58] allows easy creation of a Random Forest Classifier, which can be trained with the ground truth data. For most basic evaluation the ground truth is split into training and testing and a classifier trained on the training data is evaluated over the testing data. As Random Forests have only a few tweakable parameters they are relatively easy to use. In this thesis following parameters are tweaked for improved performance:

- n_estimators: number of random decision trees generated for the forest
- max_depth: maximum number of decisions to make in each tree to get to result
- max_leaf_nodes: maximum number of decision endpoints for the tree

After the classifier object has been trained with ground truth data, it is possible to get some information about the classifier. In this thesis the feature_importances_ data structure from the classifier object is used to get the sizes of the roles of every single feature in the detection process.

For all flows with features in a set F a classifier γ produces a vector of probabilities $\gamma(F_f) = G = [G_1, G_2, ..., G_c]$ for flow index f and class number $c \in C$. Class numbers are connected with the set of application runs A. Functions ξ and ϵ shall return the start time and end time respectively for both F and A. Thus a set of flow features which have been recorded during single application run r is defined by

$$P_r = \{\gamma(F_f) | F_f \in F : \xi(F_f) \geqslant \xi(A_r) \wedge \epsilon(F_f) \leqslant \epsilon(A_r) \wedge \epsilon(F_f) \leqslant \xi(A_r) + s\} \quad (5)$$

with s being the time threshold for an application to be considered starting. The three ways the application prediction is calculated from launches' associated flows in this thesis are shown below. The first way

$$o = \max_c \text{count}(\{\text{argmax}(G) | G \in P_r\}) \quad (6)$$

gives an equal affection to each flow prediction during application running regardless of probability distribution. Second way

$$b = \underset{c \in C}{\text{argmax}} \left\{ \sum_{G \in P_r} G_c \right\} \quad (7)$$

takes the probability distributions of each flow into account. Third way

$$h = \underset{c \in C}{\text{argmax}} \left\{ \sum_{G \in P_r^*} G_c \right\}, P_r^* = \{G | G \in P_r : \max(G) \geqslant \tau\} \quad (8)$$

discards the flows with which classifier was not confident and sums the leftover class probability distributions. In this thesis the default value 0.4 is used for τ, because a quick test showed that this way not too many flows get discarded, but the detection gets noticeably better.

6.6 Metrics used

As the predictions produced by the Random Forest algorithm are not always right, there is a need for metrics to assess the goodness of the detection. These metrics are calculated with a vector of true class indexes T, a vector of predicted class indexes P, a set of classes C and the Kroneker Delta function δ. Accuracy

$$\alpha = \frac{\sum_{i=1}^{|T|} \delta_{T_i P_i}}{|T|} \tag{9}$$

shows the fraction of predictions that were right. Precision

$$\phi = \frac{\sum_{c \in C} \frac{\sum_{i=1}^{N} \delta_{T_i P_i} \delta_{P_i c}}{\max(\sum_{i=1}^{N} \delta_{P_i c}, 1)}}{|C|} \tag{10}$$

shows how trustworthy a predictions is on average. Recall

$$\rho = \frac{\sum_{c \in C} \frac{\sum_{i=1}^{N} \delta_{T_i P_i} \delta_{T_i c}}{\max(\sum_{i=1}^{N} \delta_{T_i c}, 1)}}{|C|} \tag{11}$$

shows how good is the per class recognition on average. The maximum selections in Eq. 10 and Eq. 11 are added, because if some class has no data at all, the value for average calculation should be set to 0. Lastly F1 Score

$$f_1 = 2\frac{\phi\rho}{\phi + \rho} \tag{12}$$

is the harmonic mean of precision Eq. 10 and recall Eq. 11.

In addition to overall accuracy α, these metrics highlight the badly detected applications with low number of datapoints. Because of the low number of datapoints, these cases will have a weak effect on α, but simultaneously the effect on the rest of the metrics will be much stronger.

7 Analysis

After having enough data to analyze, the algorithms can be evaluated. This section starts with the general analysis of results on the entire dataset, which is followed by an in depth look at the actual per application recognition. Next the processing and storing requirements are evaluated by tweaking the classifier's parameters to optimize it's size. Application set size is also evaluated by trying the detection on different sized subsets. On top of that the features are evaluated to see which of them are actually important and can some of them be fully dropped from the process.

In addition to that following tests with limited conditions are introduced thus showing possibilities of utilization of these methods in real world. First detection of unknown applications is tested to see if classifier can filter the untrained applications, then cross-device detection is checked to see if every device has to have it's own training set. Lastly tests mimicking reality are performed, where firstly a very bad link is introduced to the network to see how the filtering of bad flows performs and then the detection of applications ran by hand is tested.

The main data collection for this thesis has been running for 50 days on the two devices according to the process described in Sec. 6.4. With varying technical difficulties of connection drops between the devices the data was not collected optimally. On top of that, because it was not feasible to properly filter out the cases where e.g. VPN connection had dropped, the application launches with no traffic at all have been ignored. Thus the applications which did not send data were automatically filtered out, which naturally affects statistics.

7.1 General results

Overall application detection has been evaluated over 197 apps and 132116 recorded launches on two devices. Application sets have been different because of the difference in the device OSes. The phone device had 126 apps and 52957 valid launches and the tablet device 184 apps and 79159 valid launches, where launches have at least one valid flow. Table 1 contains full test results for the whole dataset with each of the previously presented per app flow concatenation methods. The classifier used in the table can be considered as the best performing Random Forest classifier in this thesis as there are no limiting parameters set. The models are trained and tested on the same devices, thus giving better results.

The detection for the phone device performs worse than the tablet device in every test. This badness can be explained by the higher blocksize of AES, which hides actual data lengths in packets better with bigger padding. On top of padding problems the phone device might have more background processes and apps that do not close properly and continue traffic generation in the background. The phone's operating system was different and experienced more problems with keeping the VPN up for longer periods of time which negatively affected the number of valid app launches.

All of the resulting metrics are at best with the thresholded probability sums Eq. 8, but also the detected count decreases compared other detection approaches. This

likely happens because some apps do not necessarily have much traffic of their own, which allows the device's background traffic to affect the decision and because the background traffic is found in conjunction with several apps, the result becomes inaccurate.

On the other hand, as the threshold filters out uncertain flows, it is possible that no valid flows are to be found for specific application launch removing it from the statistics. Thus it is possible to compare the threshold's effect in another way, where for subindex t standing for thresholded with Eq. 8 and subindex n standing for non-thresholded with Eq. 7 a following result is seen with every device and the non-rounded data from the Table 1

$$\alpha_t \iota_t < \alpha_n \iota_n \tag{13}$$

with differences in order of per mils. This shows that if the launches that are filtered out by the threshold are added back into calculation as non-detected values, the result would actually be worse than without threshold. In other words this means that the non thresholded approach actually detects more apps right than the thresholded one. Thus it is possible to find the optimal threshold that would result in the maximum detection score, which could be especially useful if the right and wrong predictions had defined weights.

The most predicted app approach with Eq. 6 disregards the certainty values of the classifier which results in the worst performance among the three algorithms. This shows that the Random Forest's certainty values work well and should be utilized. On top of that utilization of the certainty values does not rise the workload for processing by much, making the most predicted app approach even less useful.

The precision ϕ values in the Table 1 show that the classifier's stated results per class are most often right but the recall ρ values show that less often are all of the class' instances detected as this class.

Table 1: Table of scores with 75% of the original dataset evaluated against 25% of the data four times and averaged so that every data point gets evaluated once. The default parameters for classifier were used with 10 decision trees and unlimited depth and leaf nodes. Metrics are calculated for most predicted app with Eq. 6, probability sums with Eq. 7 and thresholded probability sums Eq. 8.

Device	Eq	Detected count ι	Accuracy α	Precision ϕ	Recall ρ	F1 Score f_1
Phone	Eq. 6	52957	83%	77%	69%	71%
Phone	Eq. 7	52957	90%	82%	78%	78%
Phone	Eq. 8	50267	95%	86%	83%	84%
Tablet	Eq. 6	79159	89%	96%	86%	89%
Tablet	Eq. 7	79159	97%	98%	95%	96%
Tablet	Eq. 8	78260	98%	99%	97%	97%
Together	Eq. 6	132116	86%	92%	82%	84%
Together	Eq. 7	132116	94%	96%	92%	93%
Together	Eq. 8	128526	97%	97%	95%	96%

7.2 Per-app evaluation

To see clearer how classification works, applications are studied separately in this section. This performance evaluation can be done well with the help of confusion matrices, where x-axis represents predictions and y-axis the truth values. Thus for each combination of truth and prediction there is a point in matrix with the count of the combination's occurrence. However just plotting this matrix is not very practical

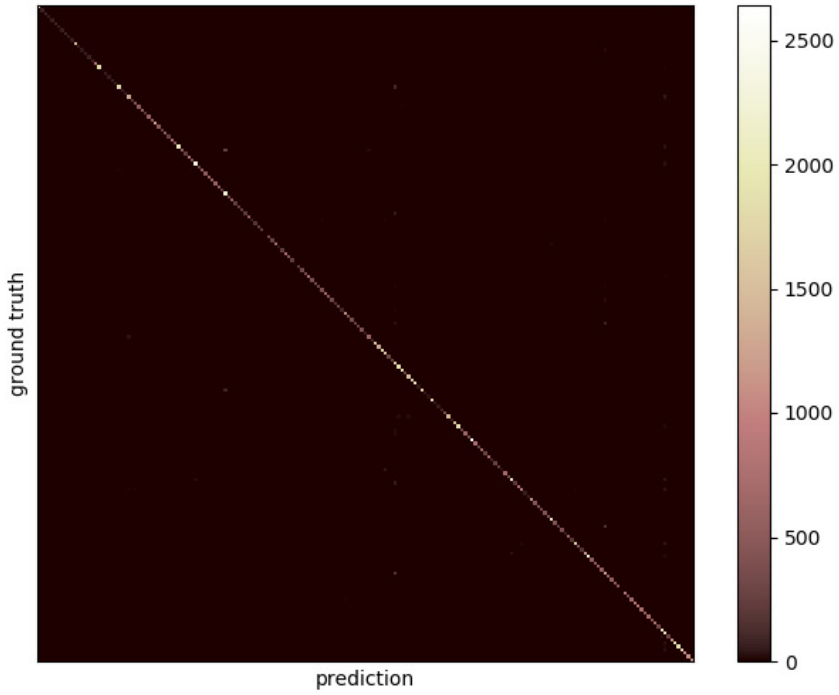


Figure 3: Confusion matrix plot without any modifications for both devices detected by using thresholded probability sums with eq. Eq. 8

as the information is not scaled as seen in Fig. 3. On top of that with this many classes one would like to zoom in and have most of the relevant cross detection information in the zoom. Next a description of how these matrices were rearranged better and an application cross detection analysis for each device shall be presented.

7.2.1 Confusion matrix arrangement

The optimization of the confusion matrices can be reformed into trying to bring all the problematic cases as close as possible to the diagonal of right predictions. As the classes are assigned indexes it is possible to optimize this assignment. Optimization requires definition of a badness metric, which in this case for class indexes vector C and a function mapping two classes to the number of occurrences for the truth and

prediction pair $\omega(t, g)$ is defined with equation

$$\beta = \sum_{t \in C} \sum_{g \in C} \omega(t, g)|t - g| \tag{14}$$

To minimize β a greedy algorithm is introduced, it finds the indexes t_w, g_w which produce the worst term of the sum and swaps indexes of classes in such a way that the classes which were indexed t_w and g_w have consecutive numbers. This is done as long as local minima is found.

After that a downwards counter of size 400 is introduced. First for 150 moves a random index pair is selected from one quarter of the best index pairs and rearranged to be closer. Then for 250 moves the original shifting logic is applied. In case the badness result gets better during counter's operation, the counter is reset and the process continues normally and in case the counter reaches 0 the process is stopped and the best found indexation of classes is returned.

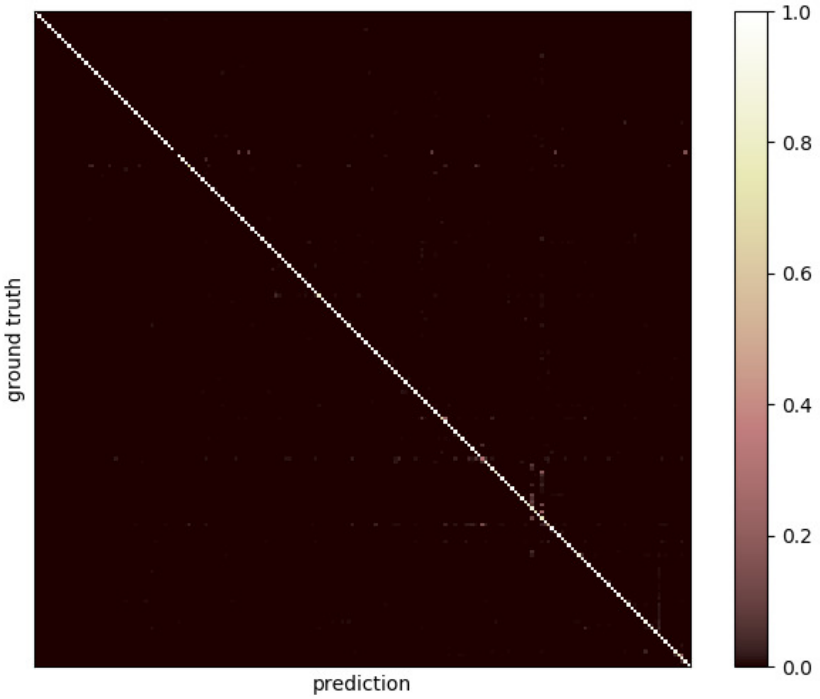


Figure 4: Sorted and normalized confusion matrix plot for both devices detected by using thresholded probability sums with Eq. 8

The values of the sorted matrix are then separately normalized over both axes so that there is one matrix with all row sums equal to one and another with all columns sums equal to one. Lastly these two matrices are summed and each value is divided by two resulting in a sorted matrix of confusions presented in Fig. 4. This

last normalization needs to be done because applications with enough datapoints still might have different numbers of the points, which would look like a bad detection in the figures. Yet on the other hand, this normalization also strongly brings forward applications which have only a few valid datapoints amplifying their data very strongly. One way to avoid this badness would be to have separate images for horizontal and vertical normalization, but this kind of doubling of plot counts would not make much sense in this scope.

7.2.2 Apps on phone device

In the phone's application detection confusion matrix in Fig. 5, it can be seen that most of the applications are detected very well. Some of the problematic applications have common mistakes with their neighbors. The vertical lines of mistakes mean that several applications are detected wrongly as one app and horizontal lines show that one application is detected as several different ones. The fact that in the Fig. 5 there are more vertical lines to be seen than horizontal, is in line with Table 1, higher precision ϕ and lower recall ρ values for phone's thresholded probability sums method.

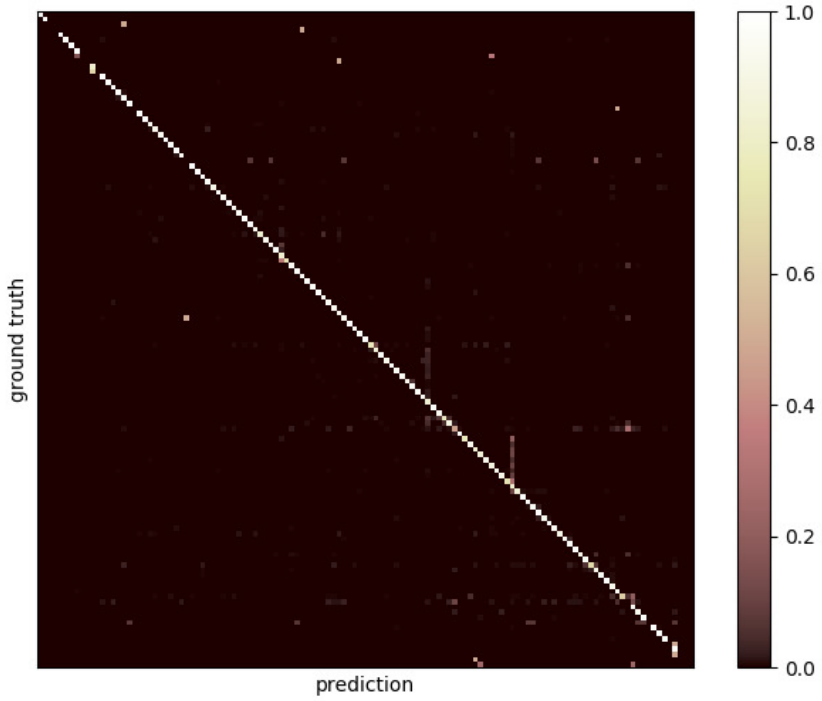


Figure 5: Sorted and normalized confusion matrix plot for phone device detected by using thresholded probability sums with Eq. 8

Some of the apps with absolutely no right hits can be seen in Fig. 6. Here "Angry

Birds Rio" are detected as "LinkedIn" and "Boom Beach" is detected as "Google Mobile". These two errors do not seem to make any sense, but in reality both of these applications had only one proper launch in the dataset. This is probably because these applications just did not need internet for functioning and thus mostly did not generate any proper flows. Thus "Angry Birds Rio" and "Boom Beach" appear only once in truth-prediction table and that one appearance time they are not trained for classification, thus being detected as some other application with similar traffic features. It is also possible that the predicted applications were not killed properly once and thus generated flows in the background during the time the misclassified applications were running.

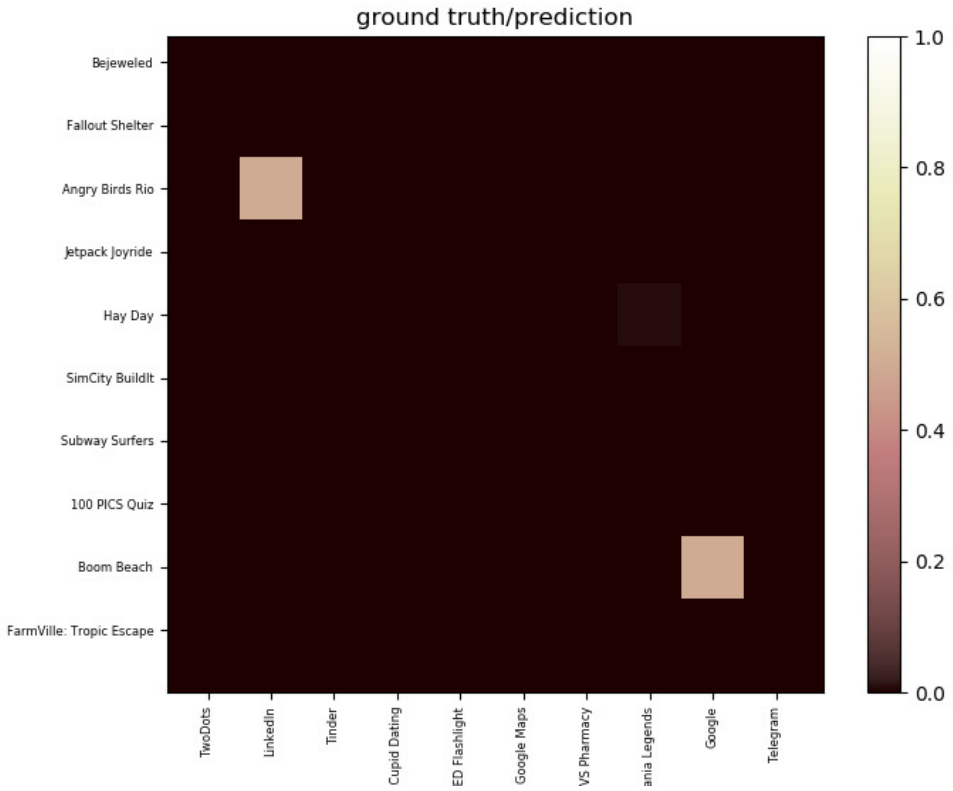


Figure 6: Sorted and normalized and zoomed confusion matrix plot for phone device detected by using thresholded probability sums with Eq. 8

In Fig. 7 lower right corner of the phone device's confusion matrix is enlarged. According to the image, the model seems to classify "Dice With Buddies" not too well, but in reality "Yahtzee With Buddies" and "Hotels.com" have only 2 and 1 datapoints respectively, which makes them non-significant. The detection problems with "Google Authenticator" are because the application does not have any network traffic, thus if valid launches with traffic were recorded, the traffic had to be from other apps running in the background. It is also possible that this traffic did not actually belong to the "Lotto Results" or "AutoRap by Smule", it might as well just be OS's background traffic which happened to be sent at the right moment.

"QR Reader" in Fig. 7 being often detected as "Sudoku by Finger Arts", "8 Ball and Pool" and "Countdown", could be for example explained by these applications having similar advertisement libraries and "QR Reader" not having much other network traffic.

Also in Fig. 7 Zynga's games "Wizard of Oz Slots Free Casino" (WZ), "Willy Wonka Slots" could also have erroneous detections with each other, as being of the same type they might run similar statistics and activity gathering algorithms as well as payment systems. On the other hand, the reason why some of WZ launches were detected as "Disney Magic Kingdoms by Gameloft", could happen because Gameloft and Zynga might have shared libraries.

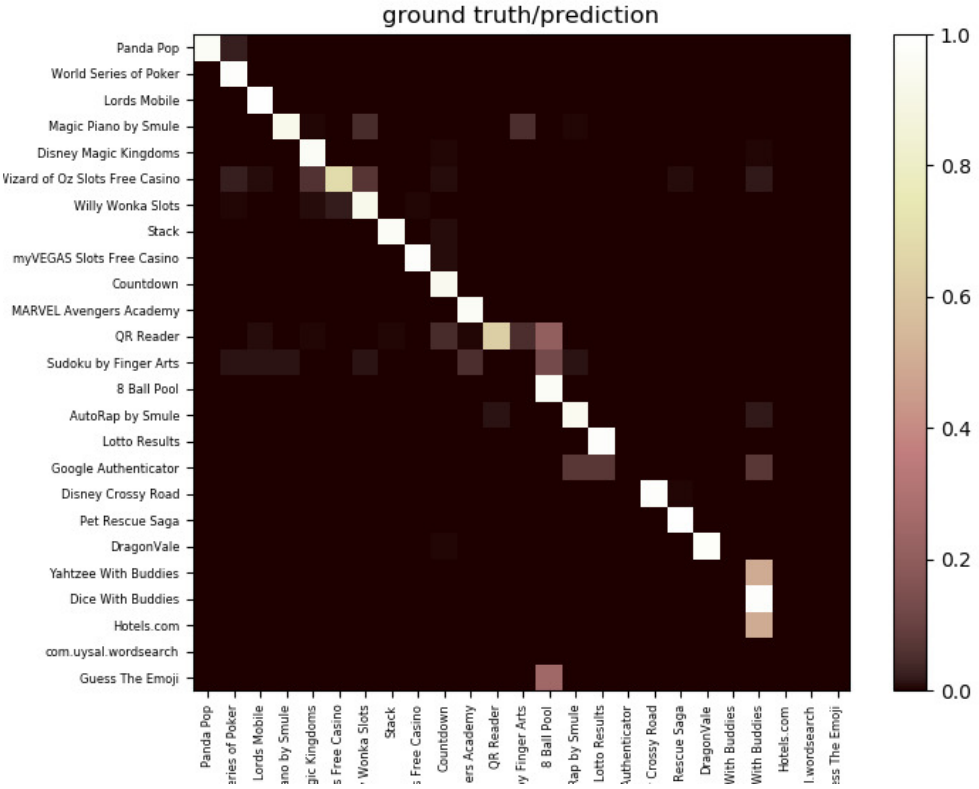


Figure 7: Sorted and normalized and zoomed confusion matrix plot for phone device detected by using thresholded probability sums with Eq. 8

In Fig. 8 we have the most confusing apps of the phone's application selection. Figure shows that several applications that do not have anything special to do with each other are detected as "DoubleDown Casino" there is no seemingly clear explanation to this. A search for explanation was done by looking at Software Development Kits (SDK) these applications use, which are shown in the Table 2. The table, however, does not help this investigation too much as the most commonly misclassified applications do not seem to use any common SDK's with "DoubleDown Casino". Thus the only explaining factor defining the exact applications being mixed up with "DoubleDown Casino" would be the fact that these applications were installed

on the phone simultaneously. On top of that the “DoubleDown Casino” does not seem to have any detection problems on the tablet, so it could have been some technical issues on the phone device, or it is also possible that the higher blocksize of the phone’s cryptoalgorithm pads in such a way that specifically these applications get mixed up.

Table 2: “DoubleDown Casino”s used SDK’s presence in applications, which had multiple detections as the DDCasino

	DDCasino predictions / Total Starts	ZenDesk	Metal	Swift SIMD	Image IO
Spotify Music	128/205				
Cookie Jam	107/1092				
Criminal Case	56/293				x
Wallpapers	48/258				
Foap - sell your photos	26/319		x	x	
Voxer	18/86		x	x	
QuizUp	16/71		x		
Kik	11/183		x	x	

7.2.3 Apps on tablet device

For the tablet device the full confusion matrix in Fig. 9 looks better, there are a few random spots around, which represent false detection cases. As these spots are very vague, they do not draw much attention, nor do they require much further investigation. Overall the tablet device was much more stable during the data generation, with most of the apps having at least 100 valid runs, the reason for this could be the different OS it was running. On top of that the shorter padding in 3DES must have helped as well.

The concentrated confusion area in Fig. 9 is enlarged in Fig. 10. Firstly we can see that many applications are being detected as “Spotify Music”, a reason for this could be the fact that Spotify, being a music player, is designed to work in the background, thus closing it does not necessarily stop all of it’s processes and it creates traffic during other applications running time. The reason why Spotify does not mix with all of the applications is that it was installed simultaneously only with a certain set of other applications and the applications that produce a lot of traffic probably overpower Spotify’s background connections.

“Sudoku by Finger Arts” and “Raven” both get detected as multiple different applications, this happens because both of them do not have much of own traffic. Yet still they also get right detections, which suggest that these applications have some kind of network traffic of their own, or that the classifier categorizes some of the background traffic as these applications.

Additionally “Secret Menu for Starbucks Free”, which has recipes for beverages, gets often detected as “Great Clips”, which is intended for reserving times at barbershops. Both of these applications are activity oriented and might therefore have

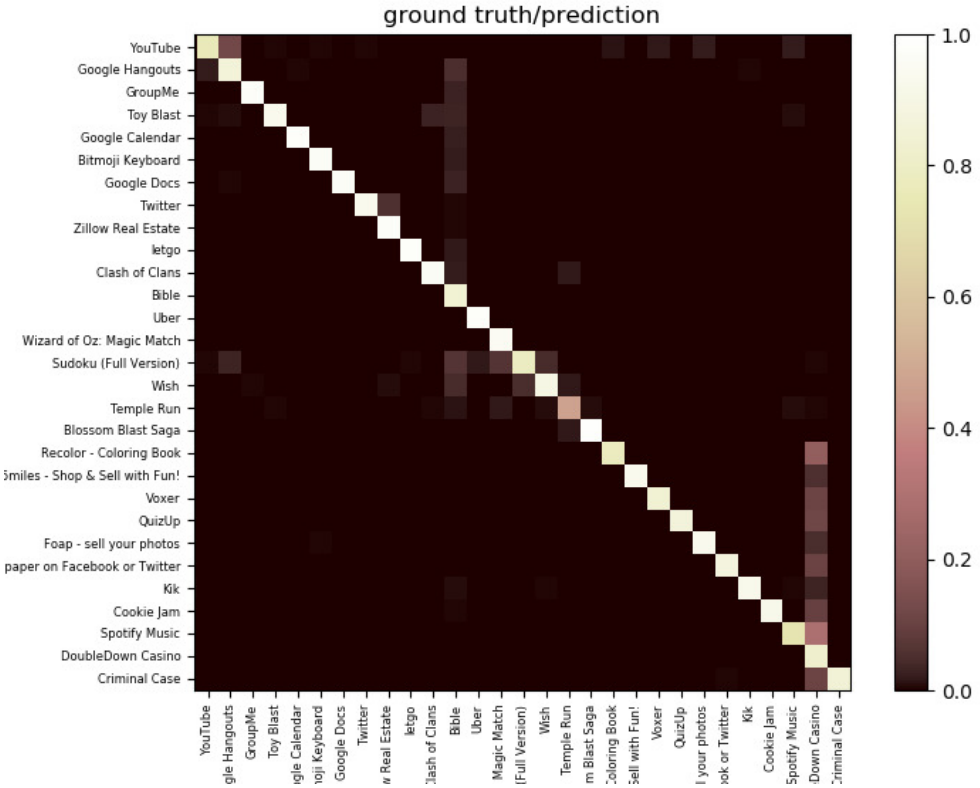


Figure 8: Sorted and normalized and zoomed confusion matrix plot for phone device detected by using thresholded probability sums with Eq. 8

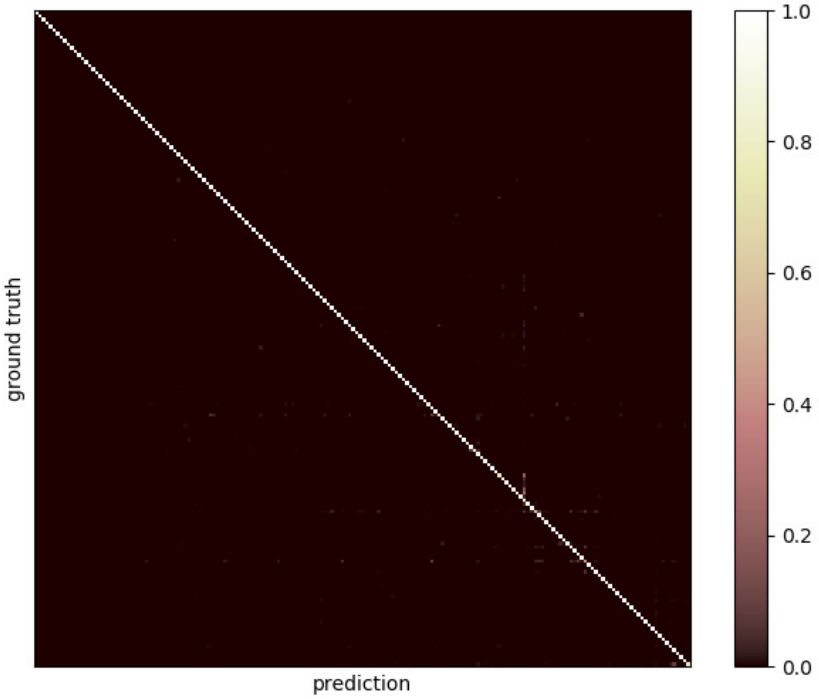


Figure 9: Sorted and normalized confusion matrix plot for tablet device detected by using thresholded probability sums with Eq. 8

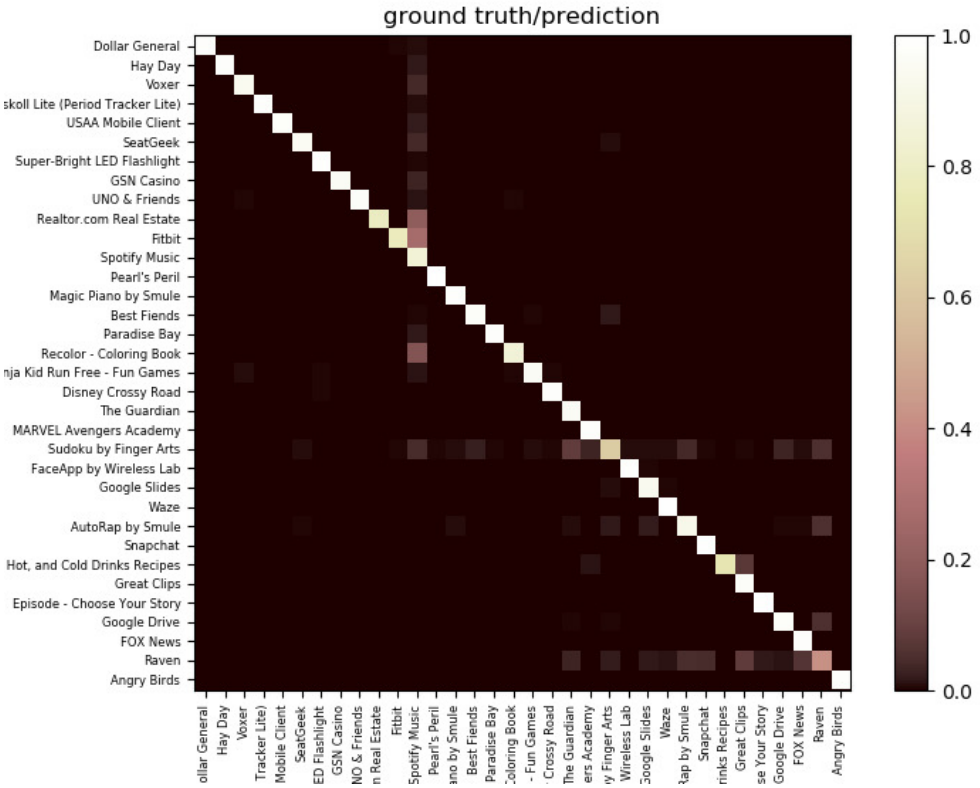


Figure 10: Sorted and normalized and zoomed confusion matrix plot for tablet device detected by using thresholded probability sums with Eq. 8

some common transmissions of data, although there are no common SDKs between these two.

7.3 Classifier size

On top of application detection quality, the training and classification speeds must be addressed as well as disk size required by the classifier as these will grow with the number of applications. The processing time for data generated on two devices in a week is parsed ready for machine learning in 18 minutes 30 seconds. The parsing was done on a Linux PC with Intel Core i7-4700MQ CPU @ 2.40GHz processor and 16 Gb of RAM. Therefore with straightforward calculation it would be possible to maintain computation of a constant flow from about 1000 devices without big modifications to the code.

However 1000 devices is not a high enough number for production environment cases, as for example it is possible that something breaks in the process, and the data has to be recalculated. The detection and training speeds are very dependent on the classifier size on disc, as bigger classifier means more evaluation steps. In addition to application count, classifier can be given limitations, and it is possible to reduce the computation required by picking up the right limiting parameters.

To assess which limits are good and what their effect is on accuracy, following plots were generated separately for phone and tablet devices, for each of the parameters described in Sec. 6.5 a set of values was evaluated. The number of trees has been tested with values between 4 and 28 with steps of 4, depth has been limited with values between 15 and 50 with steps of 5. Additionally leaf nodes have been limited with values starting from 250 and steps of 250 until 1500, where step size increases to 500 until 3000 and lastly step size of 1000 until 5000 leaf nodes. For each combination of values 10 classifiers were trained with consecutive selection of 10% of the data for testing and the rest for training. In a similar manner as for application detection calculations, these results of the 10 classifiers are then averaged into one datapoint representing the set of parameters. The probability sum approach Eq. 7 was used for detection to keep all applications and flows in the detected set.

Fig. 11a and Fig. 11b visualize effect of the number of trees on the classification accuracy. Higher number of trees helps avoiding overfitting and allows detection to be more robust. It is possible to see that in this case the risk of overfitting is not too high, thus there is no need to get more than 8 or 12 trees to get almost the best possible results. These figures also show that in case of AES encryption the number of trees affects the detection quality a bit more, which can be explained by the more monotonic data padding.

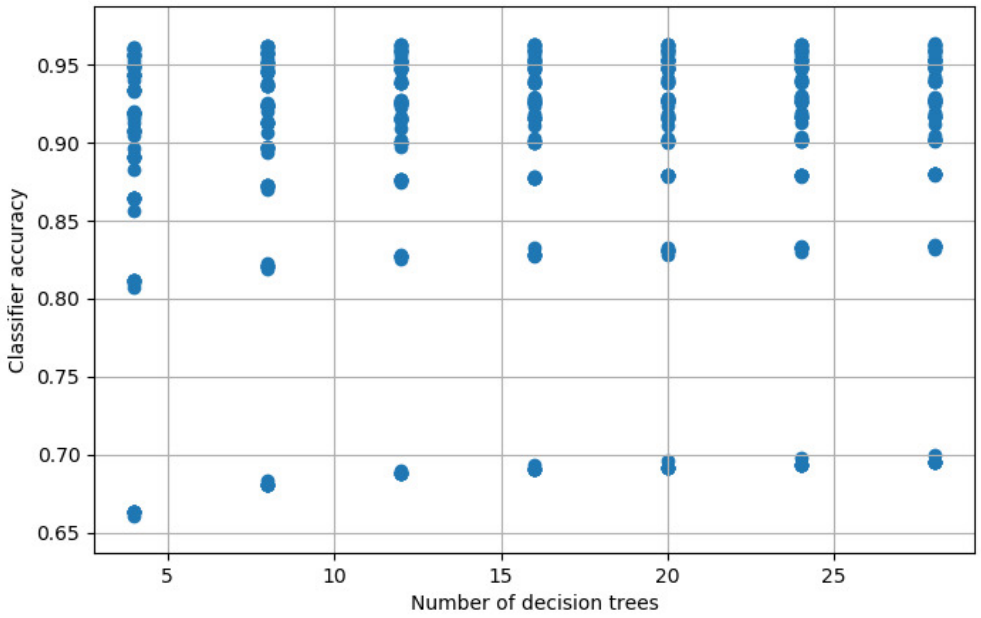
In case other parameters of the classifier are set low, Fig. 11a and Fig. 11b also show that the detection quality benefits from having more trees. Still having more than 16 trees does not raise accuracy much.

Maximum tree depth plots can be seen in Fig. 12a and Fig. 12b. These figures show, that no significant accuracy gain comes from having trees with more than 25 levels. In comparison to the number of trees, the higher depth does not give better results after a depth of 25 even when looking at the tests where all other parameters were set to low. Additionally as the mobile phone's data in Fig. 12b has fewer applications to be detected, it also gets better accuracy on equal depth limits, which suggests that the depth requirement will grow with the number of applications.

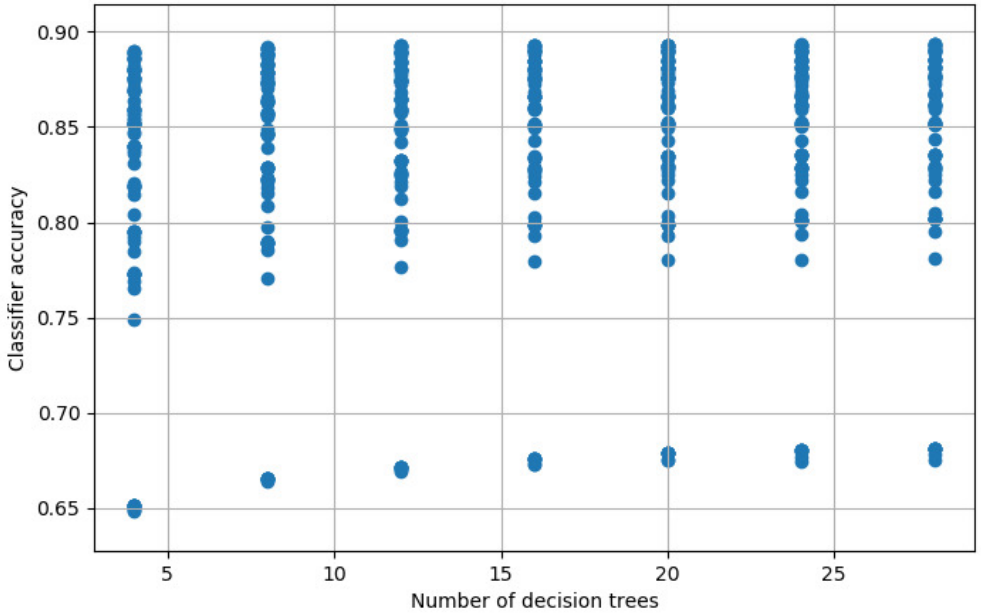
Another interesting phenomenon can be seen in the lower left part of Fig. 12a, the accuracy around the 70% is a bit higher for the lowest allowed tree depth than for other values. This can be due to decision tree algorithm's overfitting nature or just a local maxima point which produces good numbers for given datapoints but would not perform as well on any other dataset.

In Fig. 13a and Fig. 13b it can be seen that the number of leaf nodes affects the detection accuracy the most. One sign of this is that the variety in classifier's accuracy in leaf node plots is much smaller than in the plots by maximum depth and number of trees. Although this sign can be a bit misleading as for example the tree depth count could have also been measured for very low values which would surely reduce the accuracy regardless of other parameters, the accuracy is still getting better until the tests maximum node limit of 5000 is reached with both AES and 3DES algorithms.

In both cases it is possible to say that limiting leaf nodes to around 4000 yields results which cannot be made significantly better. One difference between the AES and 3DES algorithms in this regard is that with 3DES the higher leaf nodes limits

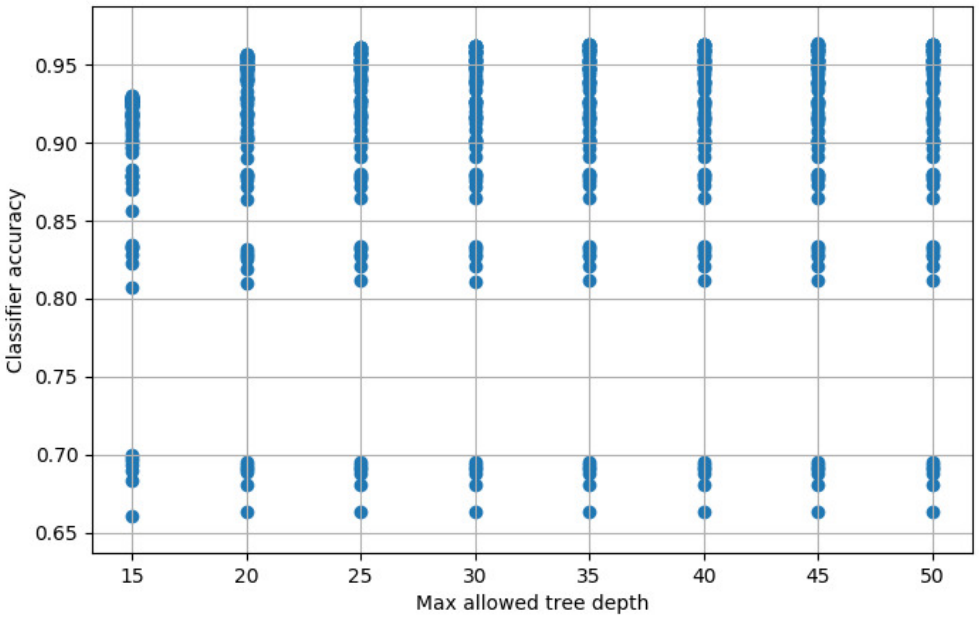


(a) Tablet device with 3DES

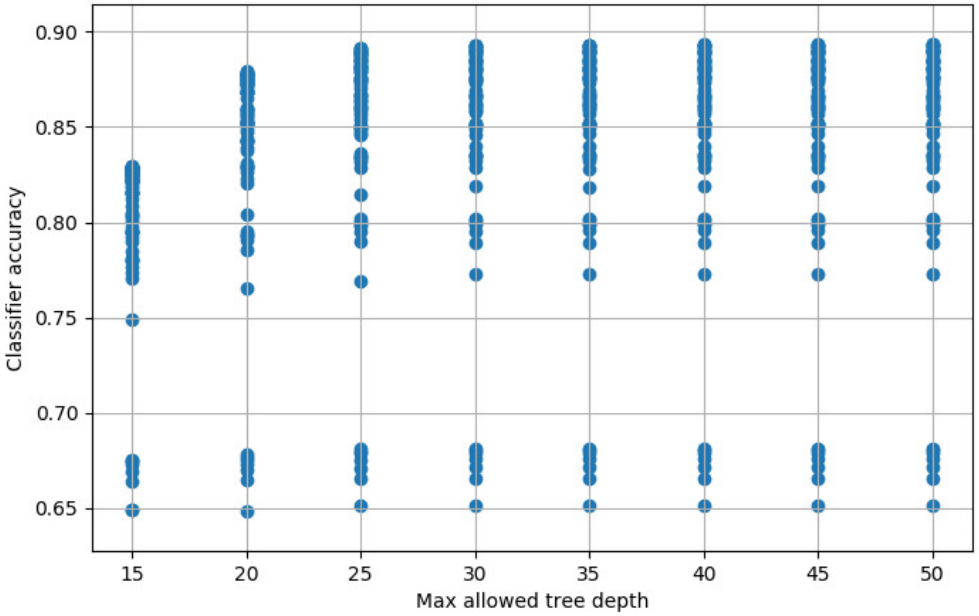


(b) Phone device with AES

Figure 11: Accuracy Eq. 9 calculated with Eq. 7 plotted against number of trees in Random Forest



(a) Tablet device with 3DES



(b) Phone device with AES

Figure 12: Accuracy Eq. 9 calculated with Eq. 7 plotted against maximum depth in Random Forest

make the results better for any set of other parameters, but in case of AES if other parameter values are limited, accuracy stops increasing already at around 2000 leaf nodes. This can possibly be explained by the smaller number of detectable applications or the bigger padding sizes.

Lastly Fig. 14a and Fig. 14b show classification accuracy in comparison to classifier size. The sizes are calculated by writing the classifier object to disc with help of a serialization library joblib for Python. As tablet device's 3DES encryption gives more variety in packet lengths and has more datapoints and applications, it's optimal size is in the ballpark of 120Mb on disc, whereas the phone device's AES detection stops getting significantly better at around 80Mb. The exact parameters for these results are to be found in Table 3.

Table 3: Optimal results and their parameters for both devices.

Device	Optimal Accuracy	Size	Depth limit	Leaf Node Limit	Trees
Phone	89%	81Mb	40	5000	8
Tablet	96%	116Mb	50	5000	8

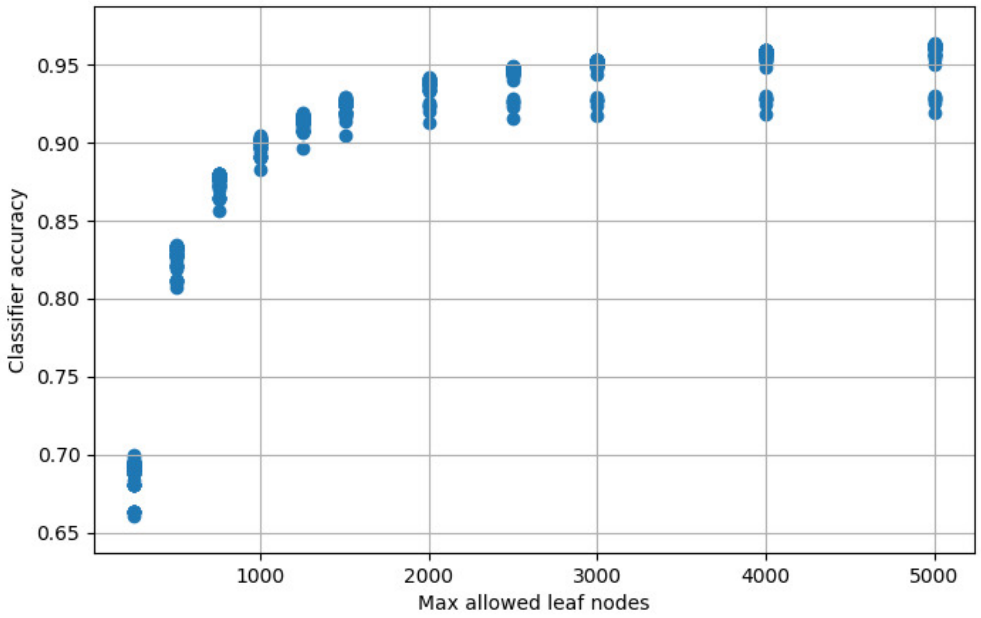
The accuracy in Table 3 is naturally not as good as the one calculated with unlimited parameters in Table 1, because the point was to minimize the classifier size. Yet considering processing and storing capacity with unlimited parameters for tablet device the classifier size is 1.2Gb which is 10 times higher than the limited version. In phone device's case the unlimited size is 420Mb which is 5 times higher than the limited classifier of the phone device. Considering that both of the unlimited classifiers had only 1% better results than the limited versions, it is clear that limiting makes sense. As a downside limiting might need to be reassessed every time new data is added, thus making it a bothersome process.

7.4 Number of applications

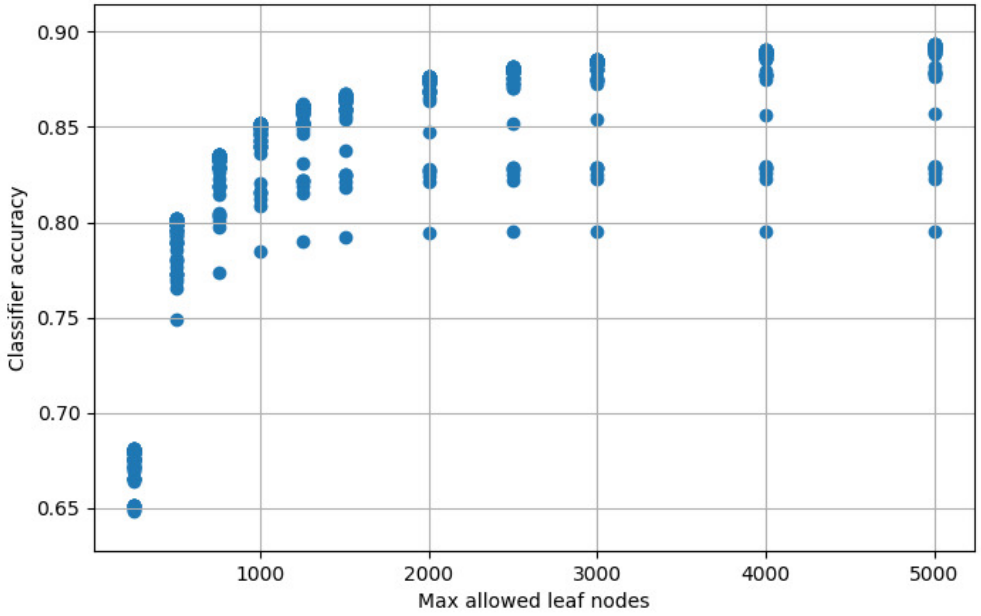
One problem is that the classifier tends to get very big without limitations, but in addition to limitations, the number of applications has direct impact on the detection quality and size. Fig. 15 shows for both devices the average accuracy for a randomly selected subsets of applications. For each size of subset the number of random selections to test is calculated for the count of apps to be selected s and number of all apps a with

$$\theta = \lfloor -20 \ln(s/a) + 1 \rfloor \tag{15}$$

which gets more samples for the smaller counts of apps. It is important to make more tests on small subsets, because the variation in results is bigger as it depends on which apps are randomly selected. For each subset 25% of datapoints were evaluated on a classifier trained on the 75% of the data four times, thus testing against each datapoint once. In case some applications have small number of datapoints, it can be that the training or testing parts of the data are empty especially when testing on 2 applications. In this cases the application set is skipped.

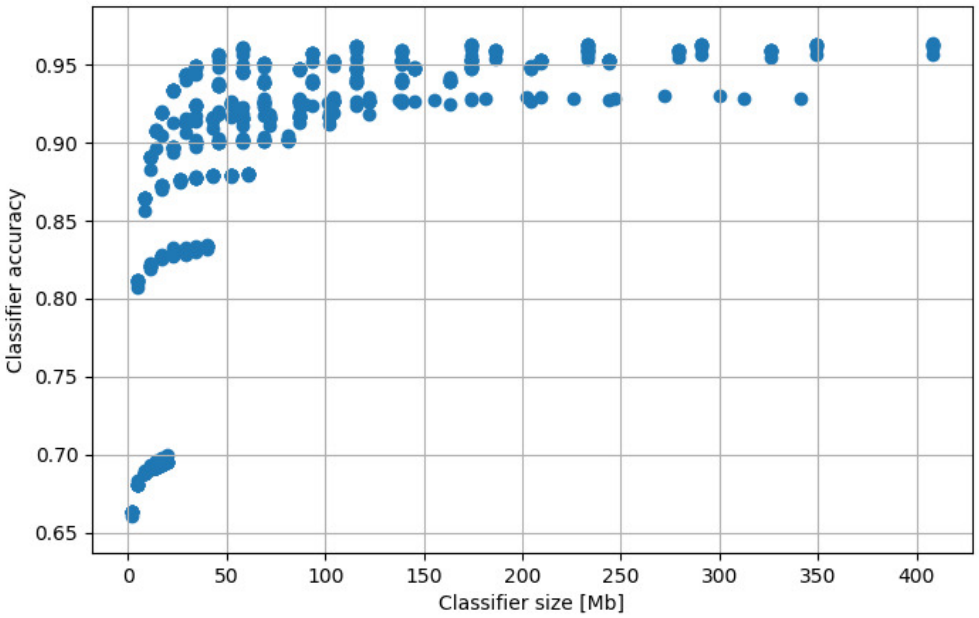


(a) Tablet device with 3DES

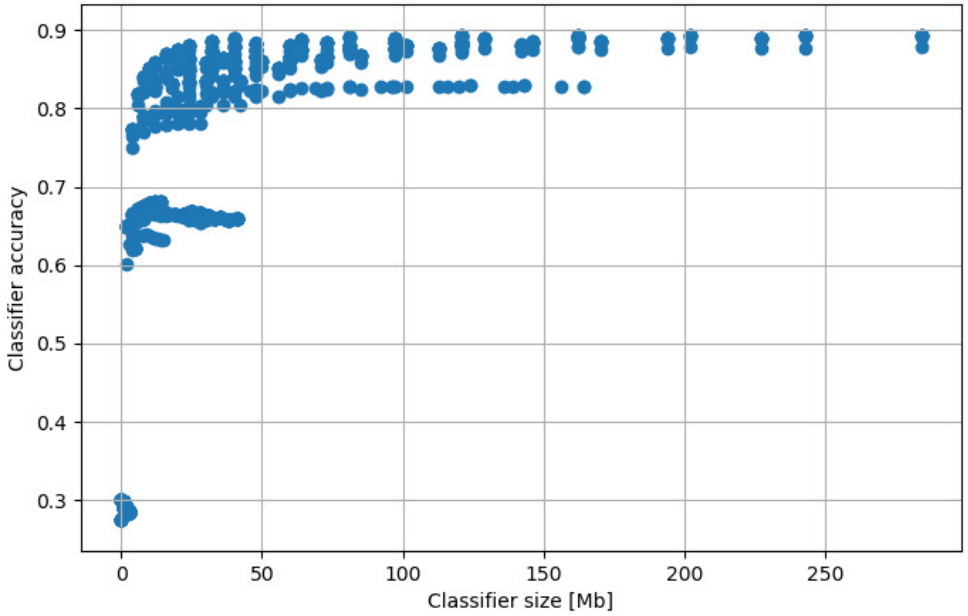


(b) Phone device with AES

Figure 13: Accuracy Eq. 9 calculated with Eq. 7 plotted against maximum leaf nodes in Random Forest



(a) Tablet device with 3DES



(b) Phone device with AES

Figure 14: Accuracy Eq. 9 calculated with Eq. 7 plotted against Random Forest classifier size after dumping with joblib

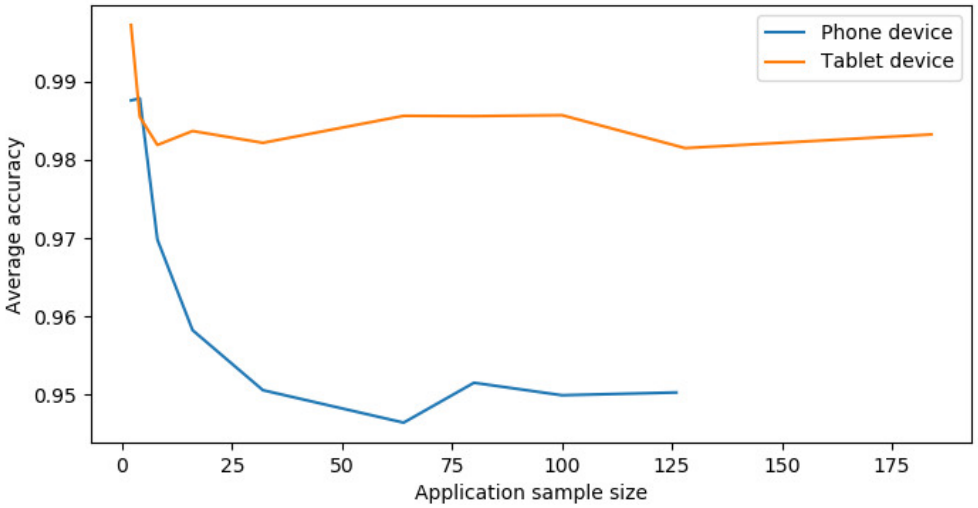


Figure 15: Accuracies calculated with Eq. 8 for both devices while training on varying sized subsets of applications.

The clearly better results for the tablet device are to be seen here as well, but in addition to that, the detection quality degradation is smaller for the tablet device, meaning that application features are better distinguishable. For phone device it's possible to see, that with 2 applications the rate of detection seems to be worse than with 4 applications. This is due to the fact that phone device has several applications with almost no data, which leads to them making the detection rate worse in cases where only two applications appear. On the other hand, in case of four applications it is probable that some of them have non-significant amount of data thus effectively having fewer detectable applications. This makes the 2 application detection rate of tablet device somehow comparable to 4 application detection rate of the phone device.

Overall with tablet device the detection rate is quite stable after selection size of 8 applications, which might suggest that raising the application count will not degrade the performance much. This can be seen on the phone device's data as well, but it is much less stable because of the applications which did not provide enough valid launches. The rise of the average accuracy at application size of 80 for the phone device can be explained by having constant majority of applications with good amount of training data.

7.5 Features

In addition to parameter tweaking the process can be optimized by dropping useless features from the data. This will not make the classifier training quicker or size smaller, but the pre-processing and amount of required data storage can be smaller.

To evaluate which features make actually sense and which not, feature importances are shown in Fig. 16. In case of tablet device the features are better distinguishable by the importance, whereas in mobile device's case the importance distribution is more monotonic which can be explained by the different padding sizes of the algorithms. The figure also shows that the device's outgoing packets are more important for distinguishing the application, which makes sense as the requests for data the applications make are expected to be more or less similar, but the content returned by the server can vary a lot. Autocorrelation of the packet lengths with small shifts also seem to be very good for application detection as they can show how many separate things are sent if only looking in one direction or how many direction switches have happened during the conversation with bigger data sent in case both directions are considered.

The badness of most of the quantile features can be explained by the fact that many flows are short or contain multiple smaller packages, thus there is no much variation in lower sizes of the packets and there are a lot of them. On the other hand, "out_len_quantile_1.0" and "out_len_quantile_0.9" seem to be very important for application detection in both devices. As 1400 bytes is the highest packet size seen in this thesis' setup and any proper data transfer will have several packets of this size one would not expect that the maximum packet sizes would matter much. Yet as the two features in question are based only on the outgoing packages, these values represent how much data the mobile device actually sends to the server. Especially

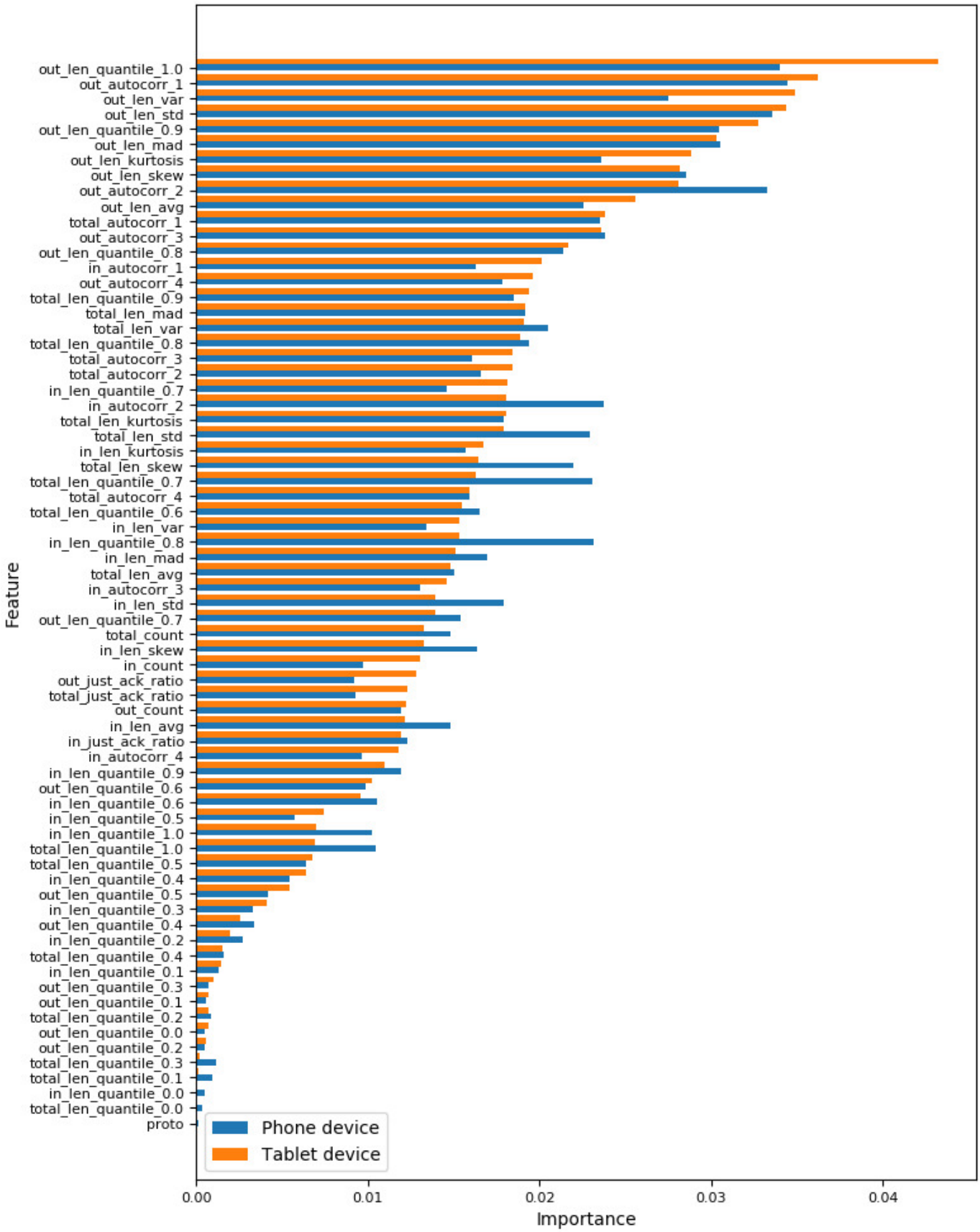


Figure 16: Feature importances of phone device and tablet device classifiers arranged by tablet devices importances. Classifiers were trained on all existing training data.

the 0.9 quantile of the outgoing packets represents clearly the possibility of the flow actually uploading some bigger things to the server. On top of that in case the application has specific patterns of data it sends e.g. statistics gathering, the maximum packet lengths may well be less than 1400 bytes and the length of one datapoint might be fixed as well, allowing to distinguish between various statistics gathering engines.

To evaluate feature importances further, a test with consecutive removal of the least important features taken from the original importance list was performed. Fig. 17 shows a plot of the effect of having fewer features on the detection accuracy. It can be seen that for both devices the decline in detection accuracy starts at 56 ignored features, thus using only 14 seems sufficient. On the other hand, the top 14 features are different for the two devices, which can be seen from Fig. 16.

The union of both device's 14 best features is of size 17, with this set of features tablet device got 97% accuracy and phone deivce 90%, which equal to scores in Table 1. The 17 features ordered by name are "in_autocorr_1", "in_autocorr_2", "in_len_quantile_0.8", "out_autocorr_1", "out_autocorr_2", "out_autocorr_3", "out_len_avg", "out_len_kurtosis", "out_len_mad", "out_len_quantile_0.8", "out_len_quantile_0.9", "out_len_quantile_1.0", "out_len_skew", "out_len_std", "out_len_var", "total_autocorr_1", "total_len_quantile_0.7". This set of features shows that direction of the packets is important as there are almost no 'total'-named features in this set. On top of that over a third of these features are autocorrelations which shows the importance packet capture order. The set also highlights that the outgoing packets are very important for detection as over 2/3 of the features are focused on them.

A check on the classifier sizes during feature removal showed that the size of the classifier on disc is relatively stable until the number of features becomes very small. When there are fewer than 10 features on phone device, the size of the classifier becomes unpredictable and can go from very low to very high. However the detection quality in this cases drops significantly and the setup does not become more optimal. Thus there is not much sense studying having very few features further.

7.6 Unknown applications

In reality it is very difficult to gather proper ground truth of all applications available on the market. Thus to be sure about classifier's results, the effect of unknown applications must be studied. Evaluation of this effect on the detection quality was done by excluding applications from the training part and appending all of the available launches of excluded applications in the test dataset. Table 4 shows accuracy results for test where 25% of the data has been selected as testing data four times to cover all the datapoints as testing points together with the excluded data. For each number of excluded applications four random exclusions were made and their values averaged over. Only the thresholded probability sums algorithm Eq. 8 is able to ignore the unknown applications and is thus used here. Other two algorithms will forcefully give a result from the training set of applications.

The ratio rows in Table 4 are calculated for count of excluded applications e and

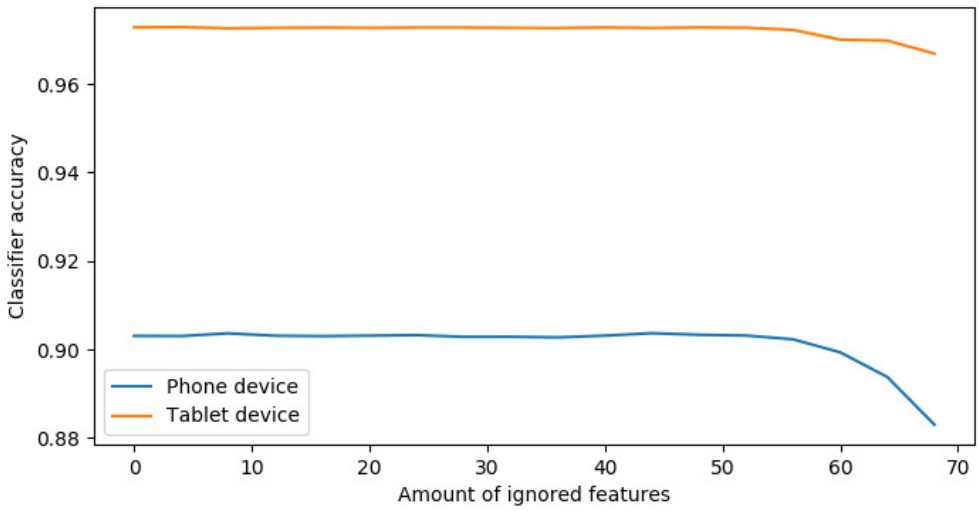


Figure 17: Classifier performance with subsets of features where least important ones have been gradually removed. 25% of the data is four times detected on a classifier trained on 75% of the data for full datapoint coverage. The detection quality evaluated with probability sums Eq. 7.

number of all applications a with equation

$$\upsilon = \frac{\frac{a-e}{4}}{\frac{a-e}{4}+e} \tag{16}$$

and approximates the relation between the trained and the untrained test data. In case every application had the same number of launches in the dataset and detection were perfect, this value would show the fraction of test data that should be detected. The division by 4 in Eq. 16 comes from the fact that a quarter of the data is used for testing purposes. Similarity of ratio values with actual accuracy suggests that the detection is not able to filter the unknown applications well enough.

The results in the Table 4 show that naturally when more unknown application launches are in the dataset for detection, the prediction quality drops. This happens mostly because the more unknown application launches are in the testing dataset, the bigger the testing dataset is all in all and if the unknonwn applications are not filtered out by the threshold, the result will be worse. On top of that some applications might have common typed flows, which would be considered uncertain by the classifier if both applications were present in the testing dataset. Thus when only one of the applications sharing same typed flows is in the training set all other applications with that flow type would be detected as the one in the training set.

Threshold values in the Table 4 clearly show that filtering out the uncertain flows has a great effect on the accuracy of detection. Compared to the $\tau = 0.4$ the $\tau = 0.9$ results are noticeably higher, especially with 64 excluded applications, where the ratio is almost doubling in favor for $\tau = 0.9$. On the downside with this high threshold values there is a possibility for right predictions to be missed as well.

Table 4: Detection accuracy with randomly excluding applications from training by excluded application count. Detection was performed with thresholded probability sums Eq. 8.

Excluded count	Threshold τ	2	4	16	64
Phone ratio υ	-	94%	89%	63%	20%
Phone accuracy	0.4	88%	89%	63%	20%
Phone accuracy	0.7	90%	89%	73%	31%
Phone accuracy	0.9	97%	97%	76%	38%
Tablet ratio υ	-	96%	92%	72%	32%
Tablet accuracy	0.4	95%	89%	71%	34%
Tablet accuracy	0.7	97%	94%	82%	43%
Tablet accuracy	0.9	99%	97%	87%	57%

7.7 Cross-device

Although having training data from all devices and OSes is more feasible than from all applications on market, it still requires a huge static workload. Next test assesses

how well the detection works when trained on one device and evaluated against another device. There were 113 applications in common between the test devices, only these applications were selected for this test, as it was not intended to detect unknown applications.

The thresholded detection algorithm was selected to minimize the error rate, as different encryption algorithms with different padding should produce different results. Results are in Table 5 and look very good, but it must noted that while detecting phone launches only 75% of launches passed the threshold filtering and for the the tablet device only 73%. Because only three quarters of application launches pass the threshold limit, it is clear that a lot of data will be lost to uncertainty, which suggest that there should be a training set for each device or at least each type of padding for encryption. On the other hand this result is much better than expected, considering the differences of the encryption algorithms for these two devices.

Table 5: Cross-device test was performed with default parameters for classifier. All available data was used for training and testing. Metrics are calculated for thresholded probability sums Eq. 8.

Train on	Evaluate on	Accuracy α	Precision ϕ	Recall ρ	F1 Score f_1
Phone	Tablet	77%	63%	59%	56%
Tablet	Phone	74%	63%	61%	57%

7.8 Bad connection

In addition to difficulties of keeping up with all devices and applications, there is a high chance of bad network connectivity, which affects the detection quality. For testing robustness against bad network conditions the mobile device connection path was changed according to Fig. 18. The controller PC creates a WIFI access point to which the mobile devices connect and routes the traffic with Network Address Translation (NAT) readdressing into network access point's WIFI connection. The readdressing allows the PC to reserve only one IP address from the network access point's IP range, which is easier than bridging the mobile devices through with their own IPs. Special filtering rules are applied to the connection marked by red arrow in such a way, that only the connection between mobile devices and VPN Server is worsened, thus leaving the control channels, where the commands are sent to mobile devices and VPN server, intact. This is implemented in Linux with traffic control program "tc qdisc" [59] and iptables routing infrastructure.

The configuration for the "bad" link is trying to mimic a very unstable connection with following parameters. Varying delay of 100ms $\pm$ 50ms is added to the link with 25% correlation between values. Packet reordering is done in the following manner: 80% of packets are sent in original order with 40% of correlation. Additionally 1% of packets are duplicated and 1% are corrupted, both with 40% correlation. On top of that a loss model is applied with two states, random loss and burst loss, where the possibility of moving from random to burst is 1% and the other way 30%.

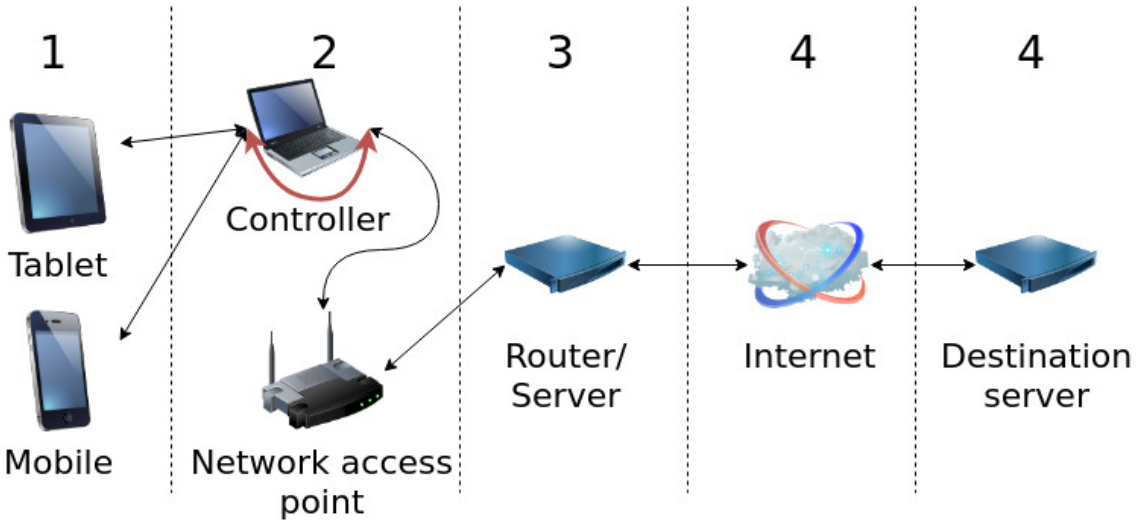


Figure 18: Connection topology for testing bad quality network connection. The red link is synthetically made bad.

During random state the loss is 0.5% and during the burst state the loss is 70%. The correlation in "tc qdisc" is stated by Beuran to mimic statistical correlation [60].

With these link parameters the devices were left running for 24 hours. Because of bad connectivity many of the application launches had no valid flows and were not counted into the statistics. The total launch count was 1953 based on launcher logs, for 21 applications on phone device and 26 applications on tablet device. The classifier detected 24 different applications for phone device and 76 different applications for the tablet device.

Classifiers were trained on all existing automatically generated launches for both devices respectively. Table 6 shows the detection metrics, which are not too good. Overall the phone device had a lot launches that did not produce any valid flows and the tablet device's launches were not detected too well. It must be noted, that some of the phone device's launches did not happen in reality, because the control of phone device was not perfect and the phone device's VPN enforcement did not seem to work well enough either. However mostly this is due to the fact that the conditions for data transfer were awful. In reality the base stations for mobile network might request data re-transmission as well as fix corrupted and out of order packets before sending them further into the internet therefore making the packets received by VPN server cleaner. As the accuracy is much higher than the other metrics in Table 6, one can say that in tablet device's case there are only a few applications that are detected well and they dominate count-wise.

Table 6: Table of scores with 1953 application launches over bad connection. The default parameters for classifier were used with 10 decision trees and unlimited depth and leaf nodes. Metrics are calculated with thresholded probability sums Eq. 8.

Device	Detected count ι	Accuracy α	Precision ϕ	Recall ρ	F1 Score f_1
Phone	85	91%	82%	77%	77%
Tablet	1398	83%	31%	27%	28%

7.9 Real usage

The last test was done against real usage of the devices without any scripting to evaluate how feasible this approach is in reality. Gathering of the test data this way requires a lot of time, thus the testing datasets are not very big. Table 7 contains the results for real usage and only launches by hand on the tablet device as well as with device where the OS is newer to evaluate how the change in the operating system and applications affects the detection quality. Classifiers were trained on all existing automatically generated launches for the tablet device, because of new OS device shared the encryption method with tablet device. Because of the same algorithms, encryption should have no effect on results and the manual usage together with newer OS should be the main problems for detection.

The Table 7 clearly shows that there is a huge degradation in detection quality with newer OS compared to the OS which was used in training. Degradation is also

a lot higher than in the cross-device test in Sec. 7.7. Noticeably higher difference in the OS version combined with much newer application versions must be one of the reasons for this. Higher threshold values also make the detection more secure but on the downside the count of applications being reported drops.

In newer OS the detection with lower threshold values has better results in the test where the device was properly used, which might suggest that although the train data has been generated by only launching applications, the detection benefits from flows that are generated during proper usage as well. On the other hand, in case of higher threshold values the detection in the no use case becomes a bit better, thus showing that there still are some flows generated by actual usage disturbing the detection process by having classifier detect them wrongly with high probabilities. But the F1 score values f_1 are worse than with actual usage in the high threshold scenario, which means that smaller part of the applications is actually detected well. This shows that actual usage does not disturb the application detection as much if a threshold is applied to ignore the user generated flows.

The detected count also drops a lot quicker with the newer device at higher threshold values, showing that the classifier's uncertainty works well. Unfortunately it still is not enough as the accuracy for the new device is very bad, which shows that it is not enough to have same encryption while training ground truth data.

Similarly in the tablet device's case the scores are a bit better when the device is in actual use and the drop of the detected application count with the increase of the threshold is also smaller. A strange thing is that in tablet device's case all of the metrics are better for the no use case, except the F1 Score. This might happen because the tested launch number is more than twice higher for the no use case than for the proper usage case, thus there is a possibility that applications which are not detected so well play a bigger role in the no use case.

Table 7: Table of scores with manual usage. The default parameters for classifier were used with 10 decision trees and unlimited depth and leaf nodes. Metrics are calculated with thresholded probability sums Eq. 8. Note that threshold $\tau = 0$ is effectively the same as maximum probability with Eq. 7. "use" means that applications were used normally without killing and "no use" means that applications were only launched and then killed from task manager. The "Newer OS"-device uses same encryption algorithm as tablet device and tablet device's classifier trained over all of the 184 apps was used.

Device	Threshold τ	Detected count ι	Accuracy α	F1 Score f_1
Tablet use	0	22	77%	69%
Tablet use	0.4	21	81%	71%
Tablet use	0.7	18	94%	92%
Tablet no use	0	45	78%	69%
Tablet no use	0.4	39	87%	81%
Tablet no use	0.7	33	94%	89%
Newer OS use	0	42	40%	25%
Newer OS use	0.4	38	45%	26%
Newer OS use	0.7	25	52%	30%
Newer OS no use	0	74	26%	15%
Newer OS no use	0.4	53	34%	19%
Newer OS no use	0.7	29	59%	28%

8 Discussion

With the results provided by Sec. 7 it is possible to understand the system better. This sections provides thoughts about the current system together with comparisons to some of the previous works and ideas for improvement.

First of all as the delta degrees are used in case of working on a sample and in this thesis' context the packet length data is full, the usage of delta degrees one in Sec. 6.3 does not actually make much sense. A study on the effect of setting delta degrees to zero should be performed, as the features it is used with could become more meaningful for flow categorization. The sole reason for using the delta degrees one is that it was a default option in the pandas library and during the implementation phase the library's operation was not studied thoroughly. Next the weak points and weird facts shall be brought forward.

8.1 Application start time

During the analysis in Sec. 7 it was noticed that setting a 20 second hard limit on the application start time in training data actually made the detection results worse. This means that the applications should have run longer than that and that the flows which end later than the 20 second mark after start of the application are important for detection. Yet still the method of just launching the applications proved to give satisfactory results. Additionally during manual detection phase the limitation on the flow length from the application start was mostly worsening the results as well, but in this case it was not so clear. This happens possibly because the flows recorded during scripted starts of the applications do not always happen at the start and additionally there could be useful flows during the application closing event.

On top of that, as the applications are very different, their starting times can vary between 1 second and 20 seconds and for some games the times can be even longer. Thus it could be possible to study every application more thoroughly to know how long does the usual start last, this could then be implemented as a varying start time for the ground truth generation and could potentially make the results more stable. There is also a possibility that some applications have constant flows that are running throughout the application's open-time, thus if the hard limit is set and the application closes a bit later, this flow gets discarded.

8.2 Preprocessing and parsing network traffic

In real use when the application is changed, it is not necessarily closed and might still for example start sending some of it's statistics reports in the background. This might interfere with the next application's detection especially if only the application starting traffic is used, therefore making the selection of the previously discussed application start time even more difficult. The results in Sec. 7.9 did not give any definitive answers to this problem, as the tests where applications were manually killed before launching next one did not perform much better.

One option to make the detection better is to try to somehow separate the flows that are generated by the OS's background processes and possibly flows that are common for several applications. Removing these flows could be done beforehand or they could be used as a separate class that would not affect the resulting predictions. This is currently implemented in the way of setting a threshold on classifier's certainty values, but if there was a way to easily separate these flows, it could potentially make the detection better. Especially in cases with unknown applications this could help a lot, as these flows would not point to wrong places.

Naturally during real usage, there will also be a lot of user generated flows which the classifier has not seen during the training phase. There is no clear way of separating these flows except for using the thresholded method similarly to the previous paragraph. This is one of the reasons start time limitation had been tried in this thesis, but it did not perform well enough.

Additionally the flows were not filtered by their length in any way in this thesis. In fact many flows might consist of only a couple of packets, making most of the proposed features meaningless. For example in work [48] Taylor et al. proposed to set a minimum limit to flow length, thus cleaning up the data which goes to machine learning. This might or might not make the detection methods used in this thesis better, because some features are also meaningful for short flows and could help the detection.

8.3 Machine learning

To utilize the gathered network traffic better this thesis used the actual timestamps of when the application was running, which in comparison to other works with similar goals helps a lot. This approach allowed concatenation of all of the flows generated by a singular application run, which improves the application detection noticeably. Just assigning flows to applications had accuracy of about 70%, but in the best case with concatenation results of 97% were achieved. Usage of Random Forest algorithm proved to be a good idea but it still could make sense to study other approaches of flow classification.

It was strange to see the classifier perform so well with the default value of 10 trees and parametrization checks revealed that even 8 trees is enough. For example in [61] Oshiro et al. showed that the optimal number of trees should be in the ballpark of 64. This suggests that the cross device results and especially the results for the newest OS could maybe be better if there were more trees in the forest. Still a quick test with Random Forest of 64 trees was done on the data for the device with newer OS with threshold $\tau = 0.4$, the results were almost the same, only the F1 score in use case got a bit better. Thus it does not make sense to create bigger forests, as the disc space and processing requirements become much bigger while the results are almost the same. One of the reasons for not needing more trees could be the fact that the per app data point count is high, which allows training each tree better.

8.4 Feature selection aspects

As noted in Sec. 8.2 there are some cases, where features of a certain datapoint are meaningless. This thesis did consider optimization of the process by cleaning up the used features and by limiting the classifier training. In Sec. 7.3 it was shown that in both cases a lot can be done, but no real tests with massive numbers of devices have been performed. In reality some of the optimizations might require constant tuning when new applications and OS versions emerge, which could become infeasible.

On the other hand as most of the traffic is transferred over TCP it could be possible to construct other features based on packet flags. TCP implements various prioritization techniques which might help detection quite a lot. For example when doing real time data exchanges like voice calls the packets might get higher priority to shorten latency between the participants of the call. Overall having this kind of information would require a more thorough study of the VPN encapsulation solutions to know if this kind of data is actually seen outside the VPN tunnel or change the capture point to see the un-encrypted data.

Additionally packet inter-arrival times could provide a lot of information about the trasmission, but they are much more difficult to parse, because of varying network conditions. Potentially some rules could be made which study the whole connection and scale the inter-arrival times according to the connection length and type. Unfortunately this was considered currently infeasible and was left out of scope of this thesis.

The approach in this thesis heavily relies on having the possibilities of traffic's separation into flows. In case all of the traffic had been sent in a singular flow these approaches would not have worked. Additionally as a method of detection's mitigation, random length data could be added to each packet by the VPN system to obfuscate the packet lengths during the transmission, which would also break the system. A singular flow VPN could also be used to block this approach, but it would probably still allow some kind of information extraction from the traffic. And the addition of random data would increase traffic amount which is not wanted by network connection providers and could become noticeably more expensive for the users as well as slow down user's the data transfers.

8.5 Robustness

To process all the capture data in a feasible time, parallelization is used with assumption that every application launch is in a separate capture file which might not be easily achieved in a real environment. On top of that the processing speed calculations do not account for the phone device's unstability. On the other hand, it could be possible to optimize the system by running Bro Network Security Monitor on the VPN server and even directly extract the features with it. This would help a lot since using tshark for faulty flow detection is slowing the process down and feature extraction with pandas is probably not the most optimal solution either.

On top of processing feasibility there are problems with training dataset generation, as shown in the Sec. 7.9, the performance of new device was far from perfect, which

creates requirements for new training data. With the constantly updated applications and OSes it is not easy to define when new training data is actually needed. Therefore a process would be needed, which constantly generates some amount of ground truth data for re-evaluation of the current classifier and then decides how much ground truth has to be generated for each application. Even if the checks and the application selections could be made automatically, the process, in it's current state, will still require some manual work to keep the devices running. This amount of manual work might make the solution unsuitable for production use-cases.

9 Conclusion

This thesis discussed utilization of information gathered around mobile devices showing that it is very rich and there are use cases ranging from preventing spread of diseases to getting very fine grained information about the device owners. As the aim of the thesis is inferring a mobile application in use by network traffic, several approaches to similar problems were discussed and the most suitable one was chosen as base.

This application detection approach was then tweaked, implemented and tested thoroughly on a 50 day ground truth dataset with labeled application launches together with their relevant network traffic from two devices. Overall the results showed that while the presented detection system is heavy and might not handle high loads, with introduced tweaks the system should be light enough to bring it into production environment and run reliably. Additionally the limiting factors were brought up giving idea of what could be done differently. However the tweaking will require a lot of work and the ground truth generation process requires more automation to be feasible for usage in modern environment, where software updates are rolling in on a weekly basis.

The results are well comparable with ones presented in Sec. 3 and the amount of different analyses has allowed better understanding of the system's performance limitations. The best achieved results were the thresholded results for tablet device shown in Sec. 7.1: 98% accuracy, 99% precision 98% recall and 97% F1 Score while still including 99% of the available launches from ground truth related to the device. This nearly perfect score shows the huge potential the presented approach has at providing reliable application detection. Additionally Sec. 7.3, Sec. 7.4 and Sec. 7.5 show that the system is not very heavy and there is room to optimize it even further without loosing much in classification quality. The cross device Sec. 7.7 and bad connection Sec. 7.8 tests gave also more or less satisfactory results, showing that the system can adapt to some extent to different devices and connection qualities.

Unfortunately the manual test Sec. 7.9 showed that although the system works more or less well on the trained device even in cases where users are utilizing the applications for real, the system is not applicable in current state to the newest versions of applications. With worst results being accuracy 26% and F1 score of 15%. Even with high thresholding the accuracy reached only 59% and F1 score only 28%, which is far from enough to call it reliable. On the other hand, the manual tests were very short and possibly a higher variety in applications could provide results with better insight. Additionally the time difference in OS and app versions was in ballpark of several years, meaning that probably if the training set consisted from newer applications, it could have much better results.

Overall the work was performed according to Sec. 4. Although at this stage still a lot of optimization work is required to be able to use the system in production, the thesis has proved that there is sense in studying these possibilities further. Additionally a more thorough study on application version differences would allow understanding the re-teaching requirements better. Which together with higher variety in OS versions would enable proper speculation of production grade deployability,

as it is at current stage impossible.

References

[1] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 566–600, Firstquarter 2018.

[2] A. Wesolowski, N. Eagle, A. J. Tatem, D. L. Smith, A. M. Noor, R. W. Snow, and C. O. Buckee, "Quantifying the impact of human mobility on malaria," *Science*, vol. 338, no. 6104, pp. 267–270, 2012.

[3] E. Enns and J. Amuasi, "Human mobility and communication patterns in Cote d'Ivoire: A network perspective for malaria control," *NetMob D4D Challenge*, pp. 1–14, 2013.

[4] K. Gavric, S. Brdar, D. Culibrk, and V. Crnojevic, "Linking the human mobility and connectivity patterns with spatial HIV distribution," *NetMob D4D Challenge*, pp. 1–6, 2013.

[5] E. Frias-Martinez, G. Williamson, and V. Frias-Martinez, "An agent-based model of epidemic spread using human mobility and social network information," in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, Oct 2011, pp. 57–64.

[6] S. Linardi, S. Kalyanaraman, and D. Berger, "Does conflict affect human mobility and cellphone usage? Evidence from Cote d'Ivoire," in *Proc. NetMob D4D Challenge*, 2013, pp. 1–3.

[7] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No NAT'd user left behind: Fingerprinting users behind NAT from NetFlow records alone," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, June 2014, pp. 218–227.

[8] A. Rao, A. Kakhki, A. Razaghpanah, A. Tang, S. Wang, J. Sherry, P. Gill, A. Krishnamurthy, A. Legout, A. Mislove, and D. Choffnes, "Using the middle to meddle with mobile," *CCIS, Northeastern University, Tech. Rep., December*, 2013. [Online]. Available: http://www.david.choffnes.com/pubs/meddle-main.pdf

[9] J. Blumenstock, G. Cadamuro, and R. On, "Predicting poverty and wealth from mobile phone metadata," *Science*, vol. 350, no. 6264, pp. 1073–1076, 2015.

[10] Y. Wang, H. Zang, and M. Faloutsos, "Inferring cellular user demographic information using homophily on call graphs," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 3363–3368.

[11] L. Hautala, "A VPN can protect your online privacy. But there's a catch," https://www.cnet.com/news/vpn-protect-online-privacy-its-complicated/, 2017, [Online; accessed 2017-September-15].

[12] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (IKEv2)," Tech. Rep., 2014. [Online]. Available: https://tools.ietf.org/html/rfc7296

[13] S. Kent and K. Seo, "Security architecture for the internet protocol," Tech. Rep., 2005. [Online]. Available: https://tools.ietf.org/html/rfc4301

[14] J. Daemen and V. Rijmen, "Specification for the advanced encryption standard (AES)," *Federal Information Processing Standards Publication*, vol. 197, 2001. [Online]. Available: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

[15] E. Barker and N. M., "Recommendation for the triple data encryption algorithm (TDEA) block cipher," 2012. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[16] S. Kent and R. Atkinson, "IP encapsulating security payload (ESP)," 1998. [Online]. Available: https://tools.ietf.org/html/rfc2406

[17] M. Dworkin, "Recommendation for block cipher modes of operation. Methods and techniques," National Inst of Standards and Technology Gaithersburg MD Computer Security div, Tech. Rep., 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[18] J. Turner, "The keyed-hash message authentication code (HMAC)," *Federal Information Processing Standards Publication*, 2008. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.198-1.pdf

[19] N. Doraswamy and C. Madson, "The ESP DES-CBC cipher algorithm with explicit IV," 1998. [Online]. Available: https://tools.ietf.org/html/rfc2405

[20] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," 2003. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[22] ——, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug 1996. [Online]. Available: https://doi.org/10.1007/BF00058655

[23] B. Cici, A. Markopoulou, E. Frías-Martínez, and N. Laoutaris, "Quantifying the potential of ride-sharing using call description records," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '13. New York, NY, USA: ACM, 2013, pp. 17:1–17:6. [Online]. Available: http://doi.acm.org/10.1145/2444776.2444799

[24] F. Girardin, F. Calabrese, F. D. Fiore, C. Ratti, and J. Blat, "Digital footprinting: Uncovering tourists with user-generated content," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 36–43, Oct 2008.

[25] K. Kumar, A. Gupta, R. Shah, A. Karandikar, and P. Chaporkar, "On analyzing Indian cellular traffic characteristics for energy efficient network operation," in *2015 Twenty First National Conference on Communications (NCC)*, Feb 2015, pp. 1–6.

[26] F. Yu, G. Xue, H. Zhu, Z. Hu, M. Li, and G. Zhang, "Cutting without pain: Mitigating 3G radio tail effect on smartphones," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 440–444.

[27] A. Espada, M. Gallardo, A. Salmerón, and P. Merino, "Performance analysis of Spotify® for Android with model-based testing," *Mobile Information Systems*, vol. 2017, 2017.

[28] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Characterizing geospatial dynamics of application usage in a 3G cellular data network," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 1341–1349.

[29] D. Naboulsi, M. Fiore, S. Ribot, and R. Stanica, "Large-scale mobile traffic analysis: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 124–161, Firstquarter 2016.

[30] T. Soikkeli and A. Riikonen, "Session level network usage patterns of mobile handsets," in *2015 13th International Conference on Telecommunications (ConTEL)*, July 2015, pp. 1–8.

[31] J. Crussell, R. Stevens, and H. Chen, "MAdFraud: Investigating ad fraud in Android applications," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: ACM, 2014, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/2594368.2594391

[32] M. Conti, Q. Li, A. Maragno, and R. Spolaor, "The dark side(-channel) of mobile devices: A survey on network traffic analysis," *CoRR*, vol. abs/1708.03766, 2017. [Online]. Available: http://arxiv.org/abs/1708.03766

[33] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""andromaly": a behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, Feb 2012.

[34] X. Su, M. Chuah, and G. Tan, "Smartphone dual defense protection framework: Detecting malicious applications in Android markets," in *2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, Dec 2012, pp. 153–160.

[35] T. E. Wei, C. H. Mao, A. B. Jeng, H. M. Lee, H. T. Wang, and D. J. Wu, "Android malware detection via a latent network behavior analysis," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, June 2012, pp. 1251–1258.

[36] M. Zaman, T. Siddiqui, M. R. Amin, and M. S. Hossain, "Malware detection in Android by network traffic analysis," in *2015 International Conference on Networking Systems and Security (NSysS)*, Jan 2015, pp. 1–5.

[37] H. Kuzuno and S. Tonami, "Signature generation for sensitive information leakage in Android applications," in *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, April 2013, pp. 112–119.

[38] N. Ruffing, Y. Zhu, R. Libertini, Y. Guan, and R. Bettati, "Smartphone reconnaissance: Operating system identification," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2016, pp. 1086–1091.

[39] X. Lu, W. Cao, X. Huang, F. Huang, L. He, W. Yang, S. Wang, X. Zhang, and H. Chen, "A real implementation of DPI in 3G network," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, Dec 2010, pp. 1–5.

[40] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "YouTube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 345–360. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068849

[41] H. F. Alan and J. Kaur, "Can Android applications be identified using only TCP/IP headers of their launch time traffic?" in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '16. New York, NY, USA: ACM, 2016, pp. 61–66. [Online]. Available: http://doi.acm.org/10.1145/2939918.2939929

[42] S. Mongkolluksamee, V. Visoottiviseth, and K. Fukuda, "Combining communication patterns & traffic patterns to enhance mobile traffic identification performance," *Journal of Information Processing*, vol. 24, no. 2, pp. 247–254, 2016.

[43] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, Jan 2018.

[44] S. E. Coull and K. P. Dyer, "Traffic analysis of encrypted messaging services: Apple iMessage and beyond," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 5–11, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2677046.2677048

[45] P. Kyungwon and K. Hyoungshick, "Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis," in *Information Security Applications*, K. Ho-won and C. Dooho, Eds. Cham: Springer International Publishing, 2016, pp. 254–265.

[46] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, Jan 2016.

[47] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 781–789.

[48] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing Android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, Jan 2016.

[49] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *2015 IEEE Conference on Communications and Network Security (CNS)*, Sept 2015, pp. 433–441.

[50] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "AntMonitor: A system for monitoring from mobile devices," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*, ser. C2B(1)D '15. New York, NY, USA: ACM, 2015, pp. 15–20. [Online]. Available: http://doi.acm.org/10.1145/2787394.2787396

[51] T. Iwai and A. Nakao, "Adaptive mobile application identification through in-network machine learning," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Oct 2016, pp. 1–6.

[52] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bigrams," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1830–1843, Aug 2017.

[53] "strongSwan project," https://www.strongswan.org/, [Online; accessed 2018-March-14].

[54] "iptables documentation," http://ipset.netfilter.org/iptables.man.html, [Online; accessed 2018-March-14].

[55] "Bro homepage," https://www.bro.org/, [Online; accessed 2018-March-14].

[56] "tshark documentation," https://www.wireshark.org/docs/man-pages/tshark.html, [Online; accessed 2018-March-14].

[57] L. Andrews and R. Phillips, *Mathematical techniques for engineers and scientists*. Spie Press, 2003, vol. 516.

[58] "scikit-learn," http://scikit-learn.org/stable/, [Online; accessed 2018-March-14].

[59] "tc documentation," http://lartc.org/manpages/tc.txt, [Online; accessed 2018-March-14].

[60] R. Beuran, *Introduction to network emulation*. CRC Press, 2012.

[61] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *Machine Learning and Data Mining in Pattern Recognition*, P. Perner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 154–168.